Computers and Operations Research 101 (2019) 250–262

Contents lists available at ScienceDirect

Computers and Operations Research

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

An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits

Ran Liu[a], Yangyi Tao[a], Xiaolei Xie[b,*]

[a] Department of Industrial Engineering & Management, Shanghai Jiao Tong University, 800 Dongchuan Road, 200240 Shanghai, China
[b] Department of Industrial Engineering, Tsinghua University, Beijing, China

ARTICLE INFO

Article history:
Received 18 October 2016
Revised 6 May 2018
Accepted 2 August 2018
Available online 11 August 2018

Keywords:
Vehicle routing
Synchronized-services
Time windows
Adaptive large neighborhood search

ABSTRACT

This paper addresses a special vehicle routing problem, which extends the classical problem by considering the time window and synchronized-services constraints. A time window is associated with each client service and some services require simultaneous visits from different vehicles to be accomplished. This problem has many practical applications such as caregiver scheduling problem encountered in the home health care industry. The synchronization constraints in this problem interconnect various vehicles' routes, making the problem more challenging than standard vehicle routing problem with time windows, especially in designing neighborhood search-based methods. A mixed-integer programming model is proposed for the problem. Motivated by the challenge of computational time, an efficient Adaptive Large Neighborhood Search heuristic is proposed to solve the problem. The approach is evaluated on benchmark instances acquired from the literature and new large-scale test instances first generated in this paper. The numerical results indicate that our solution method is able to outperform existing approaches.

1. Introduction

The vehicle routing problem (VRP) is a practical and central issue in a broad range of application systems, including distribution, transportation, healthcare, and supply chains. The classical VRP can be simply defined as the problem of designing least-cost delivery routes from a depot to a set of geographically scattered customers, subject to side constraints. In practice several variants of the VRP exist because of the diversity of operating rules and constraints encountered in real-life applications. In this paper, we focus on a particular variant of the VRP, *the vehicle routing problem with time windows and synchronized visits* (VRPTWSyn). In this problem, many vehicles start from one depot and go to service a set of geographically scattered customers such that each customer is visited only once within a given time interval (time window). More importantly, differs from the classical VRP with time windows, some customers need synchronized services, which means such customers must be served by two or more vehicles simultaneously at the same start time. Such customer is called "*synchronized services customer*" in this paper. Clearly, the synchronized service interconnects the routes of the vehicles; it makes this special vehicle routing problem more challenging to be tackled.

The VRPTWSyn has many practical applications. For example, Home Health Care (HHC) is a fast-growing medical service industry, which provides medical and paramedical services for customers at their homes. Each day the HHC company schedules the caregivers to visit and serve customers who need HHC services. The service routes for caregivers to complete services is similar to the vehicle routes in the VRP. Note in some HHC services, for example, transferring a customer with a lift between the bed and the wheel chair, the operations require more than one caregiver to be accomplished. Therefore, in such case the HHC crew scheduling problem can be formulated as the VRPTWSyn.

Clearly, the VRPTWSyn can be seen as an extension of the VRP *with time windows* (VRPTW). Because the VRPTW is a famous NP-hard combinatorial optimization problem and only relatively small instances of the VRPTW can be solved optimally within short computing times (Desaulniers et al., 2008), we focus on devising a heuristic to address the VRPTWSyn. However, the synchronization constraints make the problem extremely difficult to solve using traditional neighbourhood search heuristics. As noted by Rousseau et al. (2013), if the insertion of a special customer into a route would delay a synchronized-services customer *i*, all later customers involved in the second route (who also visits customer *i*) will also be delayed. This high number of interconnections between routes means that to insert a customer one might have to recalculate the visit time of every customer already scheduled. Constraint programming-based heuristics have been designed

* Corresponding author.
E-mail address: xxie@tsinghua.edu.cn (X. Xie).

https://doi.org/10.1016/j.cor.2018.08.002
0305-0548/

to solve this problem because constraint programming provides an easy way to express the synchronization constraints (Rousseau et al., 2003, 2013). With the goal of obtaining a high quality solution in a short time, in this paper we design special methods to address the aforementioned difficulty. Then, based on these methods, we build an Adaptive Large Neighbourhood Search (ALNS) algorithm to efficiently address this VRPTWSyn problem. Numerical experiments show that the ALNS algorithm proposed in this paper can produce satisfactory solutions within an acceptable execution time.

The rest of this paper is organized as follows. Section 2 introduces the relevant literature. A mathematical programming formulation for the VRPTWSyn is developed in Section 3. Section 4 proposes an ALNS algorithm to solve the problem. Computational results are reported in Section 5. Finally, conclusions are presented in Section 6.

2. Literature review

Compared with the standard VRP that have been the subject of intensive study for more than fifty years, there are much fewer works on the VRP with synchronization constraints. Rousseau et al. (2003, 2013) study a synchronized VRP with time windows, in which a twice-visited customer specifies precedence constraints on two visits provided by different vehicles. Constraint Programming is used in these two papers to model and solve the problem because this method provides an easy implementation path for specifying synchronization constraints. Eveborn et al. (2006) first incorporate synchronization constraints into the HHC caregivers scheduling and routing model. The synchronization defined in this work requires two caregivers to visit and service some patients at the same time. The Master's thesis by Thomsen (2006) also consider the same synchronization constraints in their HHC worker scheduling problems. Bredström and Rönnqvist (2008) extend the synchronization constraints to temporal precedence constraints between customer visits. The authors present a mathematical model for the problem and propose an optimization-based heuristic. Again, HHC is taken as an application area and three optimization objectives are considered: the first is to minimize the total travel time; the second is to maximize the sum of preferences, this is due to the fact that each client may like or dislike being served by a specific caregiver; and the third is to minimize the difference between the longest and the shortest service times among the caregivers in order to optimize the workload balance. The authors generate benchmark test instances and test their heuristic on such instances. The same authors (Bredstrom and Rönnqvist, 2007) also propose a branch-and-price algorithm for the same problem. In the root node, the synchronization constraints are relaxed and the linear model is basically a set partitioning formulation. During the solving steps, the constraints are strengthened with a branch-and-bound. The algorithm is test on the benchmark instances of Bredström and Rönnqvist (2008). Dohn et al. (2009) also propose a branch-and-price framework to solve the VRP with synchronization constraints. In their work the HHC is taken as an application background, but the test instances originating from airport operations are used for numerical experiments. Rabeh et al. (2011) address planning and scheduling two caregivers' visits in the context of HHC services, in which one patient may require separate visits from both caregivers but in a predefined order. The problem is formulated as a mixed integer programming model and is solved by the Lingo solver. Dohn et al. (2011) present a generalization of VRPTWSyn, which is called the vehicle routing problem with time windows and temporal dependencies. In addition to the standard synchronization, more general temporal dependencies are considered, such as the maximum difference and minimum difference between the starting time or ending time of two visits. A sophisticated branch-and-cut-and-price algorithm is proposed to solve the generalized problem and is tested on a set of instances derived from the well-known Solomon VRPTW benchmark. Later on, Rasmussen et al. (2012) study the routing problem under the background of HHC, in which synchronization is also extended to five types of temporal dependencies. The authors model the problem as a set partitioning problem with side constraints and develop an exact branch-and-price solution algorithm. The algorithm is tested both on real-life HHC problem instances and on generated test instances inspired by realistic settings. Haddadene et al. (2014) address the HHC routing problem with synchronization constraints using a Greedy Randomized Adaptive Search procedure, in which the search is limited in the feasible solution space and time window constraints can be checked in O(1) time by keeping some additional variables in memory. Fikar and Hirsch (2015) introduce a solution procedure for daily planning of HHC providers that operate multiple vehicles to deliver nurses to clients' homes and pick them up after service has been provided. The interdependencies, the possibility of walking to clients' locations, time windows, and assignment constraints are considered simultaneously. Afifi et al. (2016) also study the VRPTWSyn defined in this paper. The authors propose a Simulated Annealing (SA) algorithm to the problem and test their algorithm on the standard instances of Bredström and Rönnqvist (2008). Their algorithm obtains the highest quality solutions in a shorter time compared to previous approaches (Bredström and Rönnqvist, 2007, 2008). Redjem and Marcon (2016) address the HHC caregivers routing and scheduling problem, in which the visits to a patient are performed simultaneously and possibly in a predefined order. The authors design a two-stage heuristic, which sequentially tasks until all temporal constraints are satisfied. Fikar et al. (2016) propose a flexible discrete-event driven metaheuristic for the dynamic routing and scheduling problem with synchronization constraints of HHC operations. The algorithm efficiently solves HHC routing problems, facilitating trip sharing and walking in dynamic environments. Drexl (2012) presents an exhaustive survey of applications of VRP with various synchronization constraints, in which general perspectives of temporal constraints for vehicle routing problem are analysed.

In addition to the VRP and HHC-related applications, synchronization requirements are also studied for the arc routing problem (ARP). Salazar-Aguilar et al. (2012) investigate synchronized ARP for snowploughing operations in Canada where some street or road segments have multiple lanes and must be ploughed simultaneously by several snowploughing vehicles. A mixed integer programming formulation and an adaptive large neighbourhood search heuristic are proposed for this special problem. For more related synchronized ARP studies, interested readers are referred to Laporte (2016).

To summarize, some exact and heuristic algorithms have been proposed for the VRP with synchronization constraints and even more generalized temporal dependencies constraints. Because of the challenges of problem size and computational time, neighbourhood search-based methods have great potential as competitive approaches for real-life scale problem. However, only a few neighbourhood search-based methods are designed for the VRPTWSyn. Motivated by the potential applications, in this paper, we propose an efficient and effective ALNS algorithm to solve the VRPTWSyn and test its performances with existing algorithms.

3. Mathematical formulation

The VRPTWSyn is defined on a directed graph G, where $N_1 = \{1,\ldots,n_1\}$ is the set of nodes representing customers with a single service requirement, $N_2 = \{n_1+1,\ldots,\ n_1+n_2\}$ is the set of nodes representing customers who require synchronized services,

and $N=N_1\cup N_2$ representing the set of all customers. The graph $G=(V, A)$ consists of the nodes $V=\{0, n_1+n_2+1\}\cup N$ and arc set $A=\{(i, j): i, j\in V, i\neq j\}$, in which nodes 0 and n_1+n_2+1 represent the origin and destination nodes, respectively. Each customer $i\in N$ requires a service time τ_i. For each customer $i\in N$ a time window $[a_i, b_i]$ is defined, which specifies the earliest and the latest possible start service time for customer i, respectively. Each arc $(i, j)\in A$ is associated with a travel time t_{ij}. We define $t_{ii}=\infty$ for all $i\in V$ and define $t_{0,n_1+n_2+1}=0$.

Let K be the set of vehicles. The service route of each vehicle $k\in K$ initially starts at node 0 and ends at node n_1+n_2+1. Each vehicle can execute one service route and has a service duration H due to maximum work hours. We also give a unified time window $[a_0, b_0]$ to nodes 0 and n_1+n_2+1 with $b_0-a_0=H$. This represents the earliest and latest times when the vehicle may leave from and return to the depot.

A fixed cost ζ_k is incurred each time a vehicle leaves depot to serve customers. When a vehicle does not work on a particular day, the company will not pay its fixed cost. In terms of the travel costs between different nodes, it is assumed to be equal to the travel time between nodes. The total operating cost of a vehicle equals the sum of the fixed cost and travel costs. The VRPTWSyn consists of determining a set of routes that minimizes the operating costs of all the vehicles such that, (1) each customer is served by one or more synchronized vehicles according to his/her requirements, and (2) each route satisfies the time window constraints on each node.

To simplify the problem and the mathematical model, the VRPTWSyn is transformed into an equivalent problem as follows. For each customer $i\in N_2$ with two or more service requirements, we generate one or more fictive customers according to the number of his/her requirements. For example, if customer i has two requirements, then only one fictive customer i' is generated to the original customer i; customers i and i' have the same locations, service duration and time windows. In such case, we define $t_{i,i'}=t_{i',i}=\infty$, $t_{i'j'}=t_{i,j}$ for all $i, j\in N_2$ and define $t_{i'j}=t_{i,j}$ and $t_{j,i'}=t_{j,i}$ for all $i\in N_2$, $j\in V$. If customer i has three requirements, then two fictive customer i' and i'' are generated, the travel times are defined similarly, and we additional define that $t_{i',i''}=t_{i'',i'}=\infty$. We refer the set of all fictive customers to N_2'. For notation convenience, we define $V'=V\cup N_2'$ to be all the nodes and define $N'=N\cup N_2'$. Finally, we denote the set of synchronizations P^{sync}; for each pair of synchronized customers $i, j\in N'$ we let $(i, j)\in P^{sync}$. The MIP formulation for the VRPTWSyn is built as follows.

- Decision variables:

x_{ijk}: if vehicle k visits and serves customer i and travels directly to customer j, $x_{ijk}=1$; otherwise, $x_{ijk}=0$;

u_i: starting time of service for node i;

- Objective:

$$Min \sum_{i\in V'}\sum_{j\in V'}\sum_{k\in K} t_{ij}x_{ijk} + \sum_{j\in N'}\sum_{k\in K}\zeta_k x_{0jk} \quad (1)$$

- Constraints:

$$\sum_{k\in K}\sum_{j\in V'} x_{ijk} = 1 \quad \forall i\in N' \quad (2)$$

$$\sum_{j\in V'} x_{0jk} = 1 \quad \forall k\in K \quad (3)$$

$$\sum_{j\in V'} x_{j,n_1+n_2+1,k} = 1 \quad \forall k\in K \quad (4)$$

$$\sum_{j\in V'} x_{jik} = \sum_{j\in V'} x_{ijk} \quad \forall i\in N', k\in K \quad (5)$$

$$u_i+\tau_i+t_{ij}\le M\times(1-x_{ijk})+u_j \quad \forall i, j\in V', k\in K \quad (6)$$

$$a_i\le u_i\le b_i \quad \forall i\in V' \quad (7)$$

$$u_i=u_j \quad \forall (i, j)\in P^{sync} \quad (8)$$

$$x_{ijk}\in\{0, 1\} \quad \forall i, j\in V', k\in K \quad (9)$$

The objective function (1) minimizes the total operating costs of all vehicles. The first term of (1) represents the travel costs of all vehicles, and the second term is the total fixed costs of vehicles. Constraint (2) ensures all customer services are covered. Constraints (3) and (4) guarantee that each route starts from and ends at the depot. Constraint (5) ensures the flow balance of the vehicle, i.e., if a vehicle visits and serves a customer it must also leave this customer. Constraints (6) and (7) impose the time window constraints for each customer and each vehicle. Constraint (8) ensures the synchronized services. Constraint (9) guarantees that the decision variables x_{ijk} are binary.

As explained in Sections 1 and 2, the VRPTWSyn is a highly complex NP-hard problem because it covers the classical VRPTW as a special case. The above MIP formulation can be used only to solve small-scale problems. For example, we found that when the formulation was implemented in Cplex 12.6 solver for some test instances containing only 30 customers, Cplex could not provide a feasible solution even after running for 60 hours. For these reasons, we propose a metaheuristic algorithm to address large-scale problems.

4. Solution method

In this section, we describe the ALNS algorithm developed for the VRPTWSyn. The ALNS starts from an initial solution. Then, a removal method is selected to remove q customers from the current solution and an insertion method is chosen to insert them back into the current solution. The removal and insertion methods are dynamically selected by a roulette wheel mechanism according to their past performances. The new neighborhood solution yielded by these methods is accepted as the new current solution for the next ALNS iteration if it satisfies a simulated annealing-based acceptance criteria. The ALNS stops after a certain number of iterations. Compared to previous studies, our ALNS algorithm allows intermediate infeasible solutions to enhance its searching ability. Furthermore, as noted previously, because the synchronization constraints make this problem difficult to solve using traditional neighborhood search-based methods, we design special methods to tackle the interconnections among vehicle routes.

4.1. Search space and initial solution

In ALNS both feasible and infeasible solutions are allowed. A solution s is evaluated by an augmented cost function:

$$f(s)=c(s)+\alpha P_1(s)+\beta P_2(s) \quad (10)$$

where $c(s)$ is the operating cost defined by expression (1), $P_1(s)$ and $P_2(s)$ denote the total violations of the customers' time windows, the working duration of vehicles of solution s, respectively; α and β are penalization parameters. Two terms $P_1(s)$ and $P_2(s)$ are defined as follows:

$$P_1=\sum\nolimits_{i\in N'}[u_i-b_i]^+ \quad (11)$$

$$P_2=\sum\nolimits_{k\in K}[B_k-b_{n_1+n_2+1}]^+ \quad (12)$$

where x^+ represents max{0, x}, and B_k represents the arrival time of vehicle k at node n_1+n_2+1. If solution s is feasible, $P_1(s)=P_2(s)=0$ and $f(s)$ coincides with $c(s)$. According to this definition, the ALNS search process contains a mix of feasible and infeasible solutions to reduce the probability of becoming trapped in a local solution. During ALNS search process, parameters α and β self-adjust to facilitate the search space exploration. They are initialized with α_0 and β_0 and are limited within intervals (α_{min}, α_{max}) and (β_{min}, β_{max}), respectively. The values of these parameters are increased or decreased throughout the iterations. At the end of one ALNS iteration, if the incumbent solution is feasible, parameter α is divided by factor $1+\varphi_1$. If the incumbent solution is infeasible, it is multiplied by $1+\varphi_2$ while the time window constraints are satisfied, and $1+\varphi_1$ otherwise. Parameters β is adjusted according to the same rules.

4.2. Evaluation of a solution

The underlying operator of ALNS removes (destroys) and inserts (repairs) a number of customers from the current solution iteratively with the goal of improving the solution. The evaluation of a new solution is critical to the complexity of the algorithm. In this section we discuss the evaluation of a solution when ALNS removes or inserts a customer from this solution.

When a customer is inserted into or removed from the current solution s, the change in generalized cost $f(s)$ is computed to evaluate this operator. As shown in expression (10), $f(s)$ is the sum of the operating cost $c(s)$ defined by expression (1) and the penalties of constraint violations $P_1(s)$ and $P_2(s)$. Clearly, a change to $c(s)$ due to an insertion and removal move can be computed in $O(1)$ time. In terms of the time windows and working duration penalties $P_1(s)$ and $P_2(s)$, in the classical VRPTW after inserting/removing a customer into/from a route, only the start service times of the subsequent customers served later in this route should be updated (Cordeau et al., 2001). After determining the service time for each customer, $P_1(s)$ and $P_2(s)$ are obtained. However, in our problem the insertion/removal of a customer into/from a route not only changes the subsequent customers in this route, but may also change other routes because of the synchronization constraints. We propose a simple method to compute the service times of customers and the total violations of $P_1(s)$ and $P_2(s)$, as follows.

At the first step, for solution s we use a "sequence vector" g to record the service order of all the synchronized-services customers. As illustrated in Fig. 1, the top part shows a solution with three routes, k, k' and k'', in which j, l and m are synchronized-services customers. We assume their start service times are 9:00, 8:30 and 10:00, respectively. Thus, their service order is "*l-j-m*". **At the second step**, when a customer (single or synchronized-services customer) is inserted into or removed from one or two routes, we update the visit times in the modified route(s). For example, as shown in the middle part, a single service customer i has been inserted into route k between the synchronized-services customers l and j; we update the visit times of customers in route k. Note that at this step, we do not consider the interconnections caused by synchronized-services customers. For instance, customer j is visited by two routes: k and k'; now we only postpone his/her visit time to 9:30 in route k. Then, **at the third step**, based on vector g we check and adjust the visit times for each synchronized-services customer, i.e., when the visit times for a synchronized-services customer in two routes are different, we adopt the maximal one as the synchronized visit time and update the following customers in the adjusted route. For example, vector g is now "*l-j-m*"; clearly, customer l has the same synchronized start times (8:30). Next, customer j now has two different times (9:30 in k and 9:00 in k'), so his/her visit time in k' is postponed to 9:30 and the visit time for customer m in k' is also postponed to 10:30. We move on to examine the last customer in g, customer m and find out that his/her visit time in k'' must be postponed to 10:30. Now, all the visit times have been adjusted and we have completed the violation of time window constraints. Note that when a synchronized-services customer is inserted/removed into/from current solution, the sequence vector must also be updated.

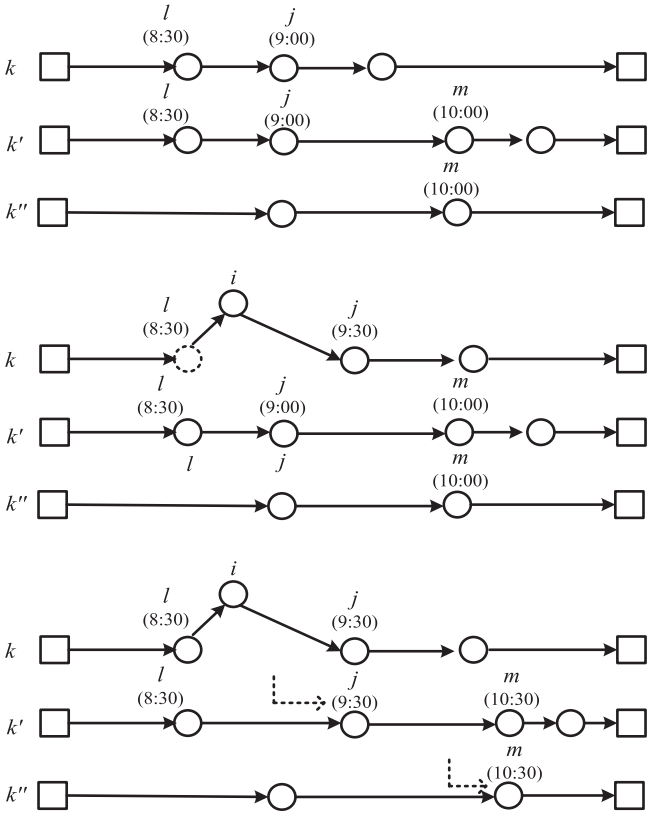

Fig. 1. Example of insertion operator evaluation in ALNS.

For the first and second steps, clearly the computation is in $O(|N|)$ time. At the third step, given that there may be a maximum of $|N_2|$ customers in the sequence vector and a maximum of $|N|$ customers' visit times are adjusted for each customer in the sequence vector, we can compute the entire time window violation in $O(|N_2|\times|N|)$ time. Although the time complexity of the evaluation of a solution is not high, we should point out that if an insertion or remove move only changes a few routes, we do not have to recalculate the other routes. One special case is when a move changes only one route, for example, inserting a single service customer when there are no interconnections among routes, we only calculate the cost of this route.

4.3. Prohibiting cross synchronization

Due to the synchronized-services requirements, in terms of an insertion operator, we must prohibit "cross synchronization" in the resulting solution. For example, suppose vehicle k visits customer i and then customer j sequentially, and vehicle k' visits customer z then w in his/her route. Let us further suppose that customers i and w represent the same customer who requires synchronized services, and that z and j also represent a synchronized-services customer. In such cases, it is impossible to visit customers i and w at exactly the same time, even if time window constraints are relaxed for customers. To avoid such cases, we use a $|N_2|\times|N_2|$ square matrix Ω to record the visit sequence among each pair of synchronized services customers. For example, as shown in Fig. 2,

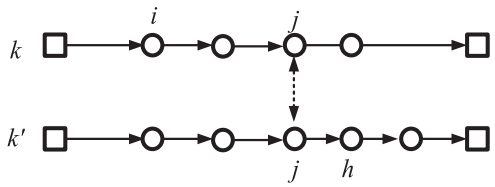

Fig. 2. Example of cross synchronization check.

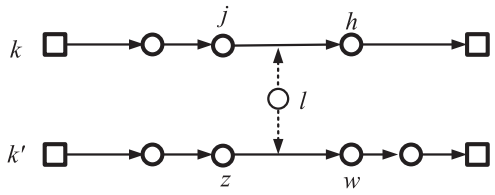

Fig. 3. Checking cross synchronization when inserting a synchronized-services customer.

in route k, customer i is visited before customer j; consequently, in matrix Ω, corresponding $\Omega[i, j]$ equals 1, otherwise it equals 0. Note that such visit sequences can be transferred between various routes. For example, in Fig. 2, suppose customer j is also visited by route k', then customer h in route k' is also regarded as being visited *after* i despite the fact that they are not visited by the same route. The corresponding value of $\Omega[i, h]$ equals 1.

4.3.1. Checking for cross synchronization

When a single service customer is inserted into the incumbent solution, we do not need to check for cross synchronization. However, when attempting to insert a synchronized-services customer into two routes simultaneously, we must ensure the insertion does not cause cross synchronization. For example, as shown in Fig. 3, consider the insertion of a synchronized-services customer l into two routes k and k'. Let us denote customer j as the first synchronized-services customer in route k preceding the insertion position of l, and h as the first synchronized-services customer in k following the insertion position of l. Similarly, let us denote z and w as the first synchronized-services customers before and after the insertion position of l in route k', respectively. We should check whether $\Omega[h, z]$ or $\Omega[w, j]$ equals 1. If either one of them equals 1, this insertion is forbidden due to cross synchronization. Otherwise, this insertion is valid.

4.3.2. Construct and update matrix Ω

For a solution s we must initialize the corresponding matrix Ω. First, we let Ω be a $|N_2| \times |N_2|$ empty matrix. Then, we construct Ω with Algorithm 1. In steps 1–6, we address each vehicle route independently, and in steps 7–17 we check the visit sequence among each pair of synchronized-services customers who may depend on two different routes. The computation time of the construction of matrix Ω is $O(|N_2|^3)$.

After a synchronized-services customer has been inserted into a solution, matrix Ω must be updated. For example, after customer l has been inserted into routes k and k', we update matrix Ω as follows. First, because in route k the immediate synchronized-services customer following customer l is h, the customers visited after h must now be visited after l. Thus, we update line l of Ω, i.e., for each column i of Ω, if $\Omega[h, i]$ is 1 we set $\Omega[l, i]$ equal 1; otherwise $\Omega[l, i]$ equals 0. Similarly, due to the visit sequence in route k', for each column i, if $\Omega[h', i]$ is 1 we also set the value of $\Omega[l, i]$ equal to 1. Next, because in route k the immediate synchronized-services customer preceding l is j, the customers visited before j must now be visited before l. We update column l of Ω, i.e., for each line i of Ω, if $\Omega[i, j]$ is 1, we set $\Omega[i, l]$ equal to 1; otherwise $\Omega[i, l]$ is set to 0. Similarly, considering the visit sequence in route k', for each line i if $\Omega[i, j']$ is 1 we also set $\Omega[i, l]$ equal to 1.

Algorithm 1
Construct matrix Ω.

```
1   Set k = 0;
2   While k ≤ |K| do
3     Select each pair of synchronized-services customers (i, j) in route k;
4     If i is served before j then Ω[i, j] = 1; otherwise Ω[i, j] = 0;
5     For each synchronized-services customers i in this route, set Ω[i, i] = 1;
6   End while
7   Do
8   For each customer i∈N2 do
9     For each customer j∈N2, i≠j do
10      If Ω[i,j] = 0 do
11        For each customer l∈N2, l≠i and l≠j do
12          If Ω[i,l] = 1 and Ω[l,j] = 1 then Ω[i,j] = 1;
13        End for
14      End if
15    End for
16  End for
17  Until Ω is not updated
```

When a removal operator has been selected and some customers are removed from the solution, we update the matrix using the construction procedure for matrix Ω.

4.4. Initial solution

We use a simple procedure to generate an initial solution. We randomly select a customer and insert this customer into the current solution at the position that minimizes the increase in the augmented cost function (10). This cheapest insertion procedure is repeated until all the customers are inserted into the solution. Note that at each step the customer can be inserted into a non-empty route or inserted into an empty route if a *free* vehicle (empty route) is still available. Obviously, we cannot guarantee the feasibility of this initial solution.

4.5. Customer removal methods

We now present the customer removal methods used in ALNS. In each removal method q customers are removed from the current solution.

4.5.1. Related removal

Our related removal is based on the "*Shaw Removal*" operator used by Ropke and Pisinger (2006). The core idea of related removal involves removing a given number of somewhat similar customers, because reshuffling related customers may improve the probability to generate feasible and better solutions. The relatedness measure of two customers i and j in this paper depends on the distance and their current visit times:

$$R(i, j) = \psi d_{i,j} + \omega |u_i - u_j| \tag{13}$$

where u_i and u_j are the visit times of two customers in the current solution; ψ and ω are weight parameters. The full related removal method is shown in Algorithm 2. A determinism parameter $p>1$ introduces randomness in the selection of the customers (line 7). Notice that after q customers have been moved from the solution, matrix Ω and vector g are then updated.

4.5.2. Worst removal

This method tries to remove customers from the current solution s whose removal results in the greatest savings. The probability of customers being selected increases with the saving values. The saving value is defined as the deviation between the augmented cost $f(s)$ when the customer is in the solution and the cost when this customer is removed. We use vector g and

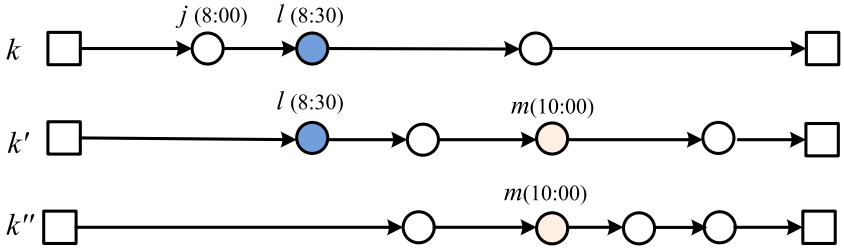

Fig. 4. Example of approximated evaluation method in worst removal.

Algorithm 2 Related removal.

```
1  Randomly select a customer i
2  Let set D={i}
3  While |D|<q do
4    Randomly select a customer j from D
5    Set the temporary customer set L to be all customers in the current
     solution not in D
6    Sort L according to the relatedness to j
7    Choose uniformly a random number y in [0,1]; set E=y^p×|L|
8    Select customer L[E] from set L and insert it into set D
9  End while
10 Remove the customers in D from the current solution
11 Update matrix Ω and vector ϑ
```

Algorithm 3 Worst removal.

```
1  While q>0 do
2    Set L represents the customers in s, sorted by the cost reduction
     obtained by removing the customer from s
3    Choose uniformly a random number y in [0,1]; set E=y^p×|L|
4    Select customer L[E] from set L
5    Remove customer L[E] from solution s
6    q=q−1
7  End while
8  Update matrix Ω and vector ϑ
```

the evaluation method described in Section 4.3 to calculate the saving values of each customer. The worst removal method is shown in Algorithm 3. Similar to the related removal method, this worst removal method also has a parameter p that determines the degree of randomization in the method (line 3).

Notice that since the saving value of each client should be calculated, and once a customer is removed from the solution many other routes may also need to be evaluated again because of the synchronization constraints, we use an *approximated evaluation method* to limit the synchronization interconnection in the computation of worst removal. For example, as shown in Fig. 4 three routes, k, k' and k'', are interconnected by two synchronized-services customers l and m. When we remove one customer j from route k and try to exactly compute the corresponding saving value, the augmented costs of both routes k' and k'' should be recalculated due to the arrival times to customers l and m and subsequence customers are changed. To reduce the computation time, our approximated evaluation method only computes the routes that are interconnect directly with this modified route k, i.e., we only update the augmented cost of routes k' and ignore the influence to route k'' because k'' is not directly interconnected by route k. Clearly, this approximated evaluation method cannot ensure an exact solution cost saving value, but it is quick and simple. Furthermore, since the customers are not strictly selected based on the saving values (a parameter p introduces randomness in the selection of the customers), such approximation does not reduce the accuracy of the whole ALNS, this observation also has been confirmed by our preliminary numerical experiments.

4.5.1. Random removal

This method randomly chooses q customers and removes them from the current solution. Similar to the related and worst removal methods, at the end of the step based on the incumbent solution we update matrix Ω and vector ϑ.

4.5.2. Synchronized-services customer removal

This method is specially designed for the VRPTWSyn. It removes all the synchronized-services customer from current solution, and empty the matrix Ω and vector ϑ.

4.5.3. Route removal

The route removal method randomly selects a route and removes all the customers in this route from the solution. When the number of customers in this route is less than q, an additional route is randomly selected and removed. This procedure is repeated until the number of removed customers is greater than or equal to q.

4.6. Customer insertion methods

We present two insertion methods in ALNS algorithm. These methods are inspired from the ones presented by Ropke and Pisinger (2006), based on best insertion and regret principles, respectively.

4.6.1. Best insertion

The best insertion method performs q times because it inserts one customer for each iteration. Each iteration consists of two steps. The first step computes the change in $f(s)$ for each previously removed customer when (1) inserting this customer into each possible position of current routes and (2) assigning one/two new vehicle route(s) to serve this single or synchronized-services customer if there is one/two vehicle(s) not been assigned to any customer. At this step, matrix Ω and vector ϑ are used to compute the insertion cost. If we cannot insert a customer into a position due to cross synchronization, we set the corresponding insertion cost equal to $+\infty$. Then, at the second step, the customer with the lowest insertion cost is inserted at the best position. Note that once a synchronized-services customer is inserted into the current solution, matrix Ω and vector ϑ are updated. This method repeatedly inserts each previously removed customer until all customers are routed.

As stated in above Section 4.2, computing the change in $f(s)$ to evaluate each customer and each insertion position needs $O(|N_2| \times |N|)$ time. Therefore, for single and synchronized-services customers, respectively, computing the first step requires $O(|N_2| \times |N|^2)$ and $O(|N_2| \times |N|^3)$ time. We use an implementation technique to speed up this step. Because at the first step we search for the minimum insertion cost for each removed customer, we can record a "*best insertion cost*" from the beginning of each removed customer, which is initialized as $+\infty$ and is updated when a smaller insertion cost is found. For a possible position, before we

complete the calculation of insertion cost, as soon as we find it has been larger than *best insertion cost*, we can directly move on to consider the next customer or position.

4.6.2. Regret insertion

The regret insertion method has been adopted by Ropke and Pisinger (2006) in their ALNS algorithm to solve the pickup and delivery problem with time windows. The basic regret method sorts the removal customers according to their *regret values*. The regret value of a customer equals the difference in cost of insertion between that customer's best insertion position and his/her second-best position. In other words, the basic regret method selects the insertion that will be regretted mostly if it is not inserted currently. The regret-k method extends the basic method by selecting the customer i such that $i = \text{argmax}_{i\epsilon}\mathcal{U}[\sum_{j=2}^{k}(\Delta f_i^j - \Delta f_i^1)]$, where $\mathcal{U}$ is the set of un-routed customers, and Δf_i^j denotes the insertion cost of customer i in the jth cheapest insertion position. Notice that different from Ropke and Pisinger (2006), we calculate the insertion cost of each customer at all possible positions within a given route, whereas Ropke and Pisinger (2006) only considered one best position within each route. Similar to the best insertion method, after a synchronized-service customer is inserted, we update matrix Ω and vector φ and re-compute the insertion positions of unplanned customers.

4.7. Selecting a removal and an insertion method

One removal and one insertion method are selected to destroy and repair the current solution. Each method i is associated with a weight w_i and a score π_i. At the beginning of ALNS, all methods have the same weights equal to 1 and scores equal to 0. Each time a removal or an insertion method is selected and a new best known solution is found, its score is incremented by the value of σ_1. When the new solution is better than the current solution and has not been accepted before, the score of this method is incremented by σ_2. If the new solution is accepted as new incumbent solution but is worse than the current solution and has never been encountered before, the score of this method is incremented by σ_3. The whole ALNS search process is divided into a number of segments, each of which consists of 100 ALNS iterations. At the end of each segment, the weights of the neighborhoods are updated as $w_{i+1} = (1-\varpi)\, w_i + \varpi\pi_i/\phi_i$ where ϕ_i is the number of times the neighborhood has been called during the last segment and ϖ is a reaction parameter controlling the inertia of the weight adjustment. During each segment the removal and insertion methods are selected independently, based on a roulette wheel selection mechanism. The probability of the ith removal/insertion method being selected is $w_i/\Sigma_{j\in\Gamma}w_j$, where Γ is the set of methods. The values of parameters $\sigma_{1\text{-}3}$ and ϖ are given in Section 5.

4.8. Acceptance and stopping criteria

A simulated annealing-based acceptance criterion is used in ALNS. More specifically, if the neighborhood solution generated by the removal and insertion methods is better than the current solution it is always accepted. Otherwise, this neighborhood solution is accepted with a probability of $e^{-(f(s')-f(s))/T}$ where $T>0$ is called the temperature, s and s' are the current and its neighborhood solutions, respectively. T is initialized as T_{start} and linearly decreased after each iteration using the expression $T = T\times\varsigma$, where $0<\varsigma<1$. From preliminary experiments we find that ALNS may become trapped in a local minimum solution when (1) the temperature T decreases to a relatively small value and (2) the solution is infeasible and has a cost less than the cost of the best known feasible solution. Therefore, when the ALNS is trapped in an infeasible solution and cannot break free during the next θ iterations, the next neighborhood solution is accepted as a new incumbent solution even if it is not an improvement.

5. Computational experiments

In this section, we conduct several sets of experiments to assess the performance of the ALNS in terms of solution quality and computation time. We generate tuning instances and discuss the setting of algorithmic parameters. We further test and compare ALNS with existing algorithms from the literature on benchmark instances. All the algorithms presented in this paper are coded in C++. The algorithm is tested on an Intel E5-2670 CPU clocked at 2.6 GHz and 2 GB memory running a Linux system.

5.1. Tuning instances

First, we derive various scaled of tuning instances from existing VRPTW benchmarks (Solomon, 1987; Gehring and Homberger, 1999). We select 6 Solomon's VRPTW instances. For each one we choose 30 customers as our customers. Among 30 customers we select $a\%\times 30$ as synchronized-services customers. For each instance we set $a=10$, 30 and 50 to achieve three test instances. If the basic Solomon instance is C102 and $a=10$ then it is denoted as C102-30-3. All such instances are called small-scale instances. Based on a similar procedure, we selected 6 instances and used all 100 customers. We let $a=5$, 10 and 20 to generate 18 total moderate-scale test instances. Finally, we select 6 instances from benchmarks of Gehring and Homberger (1999), each of which has 200 customers. Such instances undergo above procedure (using $a=5$, 10 and 20) generating 18 large-scale instances.

5.2. ALNS parameter setting

The ALNS relies on a set of correlated parameters. The most parameters settings are summarized in Table 1. In terms of parameters T_{start} and ς (Section 4.8), we chose T_{start} value such that the first neighborhood solution can be accepted with a probability of 50% if its cost is 5% larger than current solution. Parameter ς is selected such that the temperature at the last ALNS iteration is equal to $0.2\%\times T_{start}$.

Among all the parameters q is a key parameter to both algorithmic speed and accuracy. We used another parameter q' to control q: $q = \lfloor q'\times|N|\rfloor$ where q' is the percentage of customers removed in each iteration. In order to give a sensitivity analyze to this parameter, five different intervals are tested for q', namely [5%, 15%], [15%, 30%], [5%, 30%], [15%, 50%] and [30%, 50%]. At each iteration, q' is selected uniformly at random in corresponding interval. ALNS using these intervals are called "ALNS-5-15″, "ALNS-15-30″, "ALNS-5-30″, "ALNS-15-50″, "ALNS-30-50″, respectively. Tables 2–4 display the detailed results of these versions of the ALNS algorithms for different scales of test instances, including the best and average solution costs (sub-columns "*Best*" and "*Avg*") and its average time for one algorithmic run in seconds (sub-column "*Time*"). Boldface indicates the best known solution cost found in such experiments.

As shown in Tables 2–4, as parameter q increases, the run time of ALNS algorithm also increases. For example, solving the same test instance C106-100-5, the average solving times of ALNS-5-15, ALNS-15-30, and ALNS-30-50 are 17.3 s, 47.3 s and 102.1 s, respectively. More importantly, we observe that the solution quality does not always improve as parameter q increases. For three sets of instances, the solutions of algorithms with relatively high values of q, i.e., ALNS-15-30 (q' in 15–30%), ALNS-15-50 (15–50%), and ALNS-30-50 (30–50%), are better than those with small q values, i.e., ALNS-5-15 and ALNS-5-30. However, the solution quality does not increase with q among ALNS-15-30, ALNS-15-50 and ALNS-30-50. For example, consider the third set of instances with 200

Table 1
Parameter setting in the experiments.

Symbol	Explanation	Value
α_0, α_{min}, α_{max}	Initial, minimum and maximum values of α	10, 0.001, 10,000
β_0, β_{min}, β_{max}	Initial, minimum and maximum values of β	10, 0.001, 10,000
φ_1, φ_2	Parameters for updating α, β	0.1, 0.05
ψ, ω	Weight parameters in related removal	0.5, 0.5
p	Randomness parameter in related and worst removal	5
θ	Number of consecutive unimproved iterations to update the current ALNS solution	10
σ_1, σ_2, σ_3, ϖ	Parameter for selecting removal and insertion methods	33, 9, 13, 0.1

Table 2
Computational results on 30-customer test instances.

Instance	ALNS–5–15			ALNS–5–30			ALNS–15–30			ALNS–15–50			ALNS–30–50		
	Best	*Avg*	*Time(s)*	*Best*	*Avg*	*Time(s)*	*Best*	*Avg*	*Time(s)*	*Best*	*Avg*	*Time(s)*	*Best*	*Avg*	*Time(s)*
C102_30_3	**287.12**	287.12	0.3	**287.12**	287.12	0.5	**287.12**	287.12	0.6	**287.12**	287.12	0.8	**287.12**	287.12	1.0
C102_30_9	**373.01**	373.01	0.8	**373.01**	373.01	1.0	**373.01**	373.01	1.1	**373.01**	373.01	1.5	**373.01**	373.01	1.8
C102_30_15	**391.21**	391.4	1.5	391.21	**391.21**	1.8	**391.21**	391.21	1.9	391.21	**391.21**	2.4	**391.21**	391.21	3.0
C203_30_3	**301.47**	301.47	0.3	**301.47**	301.47	0.4	**301.47**	301.47	0.5	**301.47**	301.47	0.7	**301.47**	301.47	0.9
C203_30_9	345.15	346.79	0.6	343.1	347.35	0.7	345.15	349.33	0.9	**343.1**	348.41	1.1	345.15	350.38	1.5
C203_30_15	423.17	448.36	1.0	**422.74**	450.64	1.2	**422.74**	442.48	1.4	**422.74**	435.08	1.8	**422.74**	434.38	2.3
R104_30_3	**519.15**	519.15	0.3	**519.15**	519.15	0.4	**519.15**	519.15	0.6	**519.15**	519.15	0.8	**519.15**	519.15	1.1
R104_30_9	**666.71**	666.71	0.8	**666.71**	668.37	1.1	**666.71**	666.71	1.2	**666.71**	666.71	1.6	**666.71**	666.71	2.1
R104_30_15	**740.33**	747.18	1.3	**740.33**	743.92	1.5	**740.33**	744.99	1.8	**740.33**	741.74	2.3	**740.33**	740.8	3.3
R205_30_3	530.64	533.62	0.3	**528.17**	531.22	0.4	**528.17**	531.38	0.5	**528.17**	531.95	0.7	**528.17**	530.4	0.9
R205_30_9	687.83	688.96	0.7	687.83	688.02	0.9	**687.51**	687.97	1.0	**687.51**	687.8	1.5	**687.51**	687.8	1.9
R205_30_15	**745.36**	771.68	1.2	747.01	765.5	1.5	745.91	757.67	1.7	**745.36**	757.02	2.1	**745.36**	755.89	2.7
RC103_30_3	**603.82**	604.02	0.3	**603.82**	603.82	0.4	**603.82**	603.82	0.5	**603.82**	603.82	0.8	**603.82**	603.82	1.1
RC103_30_9	**757.05**	757.05	0.7	**757.05**	757.05	1.0	**757.05**	757.05	1.2	**757.05**	757.05	1.6	**757.05**	757.05	2.3
RC103_30_15	**843.63**	843.63	1.1	**843.63**	843.63	1.4	**843.63**	843.63	1.6	**843.63**	843.75	2.1	**843.63**	843.71	3.1
RC207_30_3	**552.45**	556.17	0.3	**552.45**	553.05	0.4	**552.45**	553.46	0.5	**552.45**	552.45	0.7	**552.45**	552.45	1.0
RC207_30_9	689.71	714.31	0.7	689.71	704.56	0.8	**687.2**	699.82	1.0	**687.2**	701.27	1.4	**687.2**	697.48	1.9
RC207_30_15	800.18	815.52	1.0	**796.92**	805.84	1.3	**796.92**	803.89	1.4	**796.92**	807.31	1.9	**796.29**	802.26	2.5
Avg	**569.89**	575.90	0.7	**569.52**	574.16	0.9	**569.42**	573.01	1.1	**569.28**	572.57	1.4	**569.35**	571.95	1.9

Table 3
Computational results on 100-customer test instances.

Instance	ALNS-5-15			ALNS-5-30			ALNS-15-30			ALNS-15-50			ALNS-30-50		
	Best	*Avg*	*Time(s)*	*Best*	*Avg*	*Time(s)*	*Best*	*Avg*	*Time(s)*	*Best*	*Avg*	*Time(s)*	*Best*	*Avg*	*Time(s)*
C106_100_5	**1039.96**	1041.36	17.3	**1039.96**	1040.05	32.7	**1039.96**	1039.96	47.3	**1039.96**	1039.96	71.2	**1039.96**	1039.96	102.1
C106_100_10	**1146.47**	1146.96	30.7	**1146.47**	1146.47	53.0	**1146.47**	1146.47	74.6	**1146.47**	1146.47	108.0	**1146.47**	1146.47	150.1
C106_100_20	1313.20	1319.20	66.7	1313.2	1315.82	106.5	**1310.21**	1315.14	139.4	**1310.21**	1314.35	196.4	**1310.21**	1312.19	287.6
C201_100_5	**773.11**	773.17	13.8	**773.11**	773.11	21.4	**773.11**	773.11	34.1	**773.11**	773.11	57.0	**773.11**	773.11	75.2
C201_100_10	**850.81**	850.81	26.5	**850.81**	850.81	38.6	**850.81**	850.81	56.7	**850.81**	850.81	87.7	**850.81**	850.81	111.8
C201_100_20	**940.99**	940.99	43.0	**940.99**	940.99	69.1	**940.99**	940.99	81.4	**940.99**	940.99	119.8	**940.99**	940.99	152.5
R110_100_5	1168.55	1182.50	11.8	**1161.42**	1172.51	23.5	**1161.42**	1170.72	35.3	1168.42	1175.47	58.2	**1161.42**	1171.20	76.1
R110_100_10	**1222.08**	1237.21	26.5	1226.47	1230.59	45.0	**1222.08**	1230.26	62.2	1226.92	1241.87	99.4	1234.61	1244.49	131.0
R110_100_20	1375.58	1409.67	55.1	1398.03	1414.25	81.8	1398.93	1407.72	119.2	**1372.26**	1413.10	176.4	1400.08	1419.21	237.0
R209_100_5	945.48	961.62	13.5	939.60	949.46	26.4	**937.40**	951.64	35.9	**937.40**	945.42	54.8	940.67	947.16	77.1
R209_100_10	1013.98	1022.73	16.5	**1003.72**	1014.49	29.6	1004.29	1013.39	41.1	1007.07	1016.56	63.8	1012.20	1015.58	92.5
R209_100_20	1167.27	1182.08	30.3	1167.34	1181.98	48.0	**1157.87**	1179.76	67.2	1158.29	1176.11	99.1	**1157.87**	1178.99	142.6
RC101_100_5	**1762.22**	1767.95	13.8	**1762.22**	1765.61	27.9	1765.06	1765.59	39.4	1763.69	1765.97	56.4	1765.06	1765.76	77.0
RC101_100_10	1917.3	1919.89	35.3	1891.31	1910.84	59.8	**1891.49**	1905.44	81.6	1893.01	1901.74	119.4	**1891.49**	1900.28	160.6
RC101_100_20	2103.62	2122.56	107.7	2103.62	2116.94	165.2	**2096.26**	2106.84	213.0	2101.43	2114.01	299.4	2102.59	2109.02	394.8
RC205_100_5	1309.32	1321.47	15.2	1306.74	1318.27	29.6	**1301.23**	1315.03	41.7	1304.37	1313.96	63.2	1301.42	1309.86	77.8
RC205_100_10	1378.27	1398.80	25.2	**1376.32**	1391.2	43.6	**1376.32**	1387.93	60.6	1376.34	1385.20	90.3	**1376.32**	1381.49	115.4
RC205_100_20	1595.57	1627.02	38.7	1598.99	1622.21	61.1	**1571.22**	1602.68	83.6	1573.91	1603.30	121.3	1592.08	1608.44	169.2
Avg	1279.10	1290.33	32.6	1277.80	1286.42	53.5	1274.73	1283.53	73.0	1274.70	1284.36	107.9	1277.63	1284.17	146.1

customers, as shown in Table 4, ALNS-15-30 and ALNS-15-50 are able to find 8 and 6 best solutions out of 18 test instances respectively, and ALNS-30-50 with the largest value of q finds 7 best instances. In terms of the mean value of the best solutions found by ALNS-15-30, ALNS-15-50 and ALNS-30-50, out of these 18 test instances the result of ALNS-15-30 (3264.04) is slightly better than that of the ALNS-30-50 (3274.31) and ALNS-15-50 (3265.69). We also have done the experiments in which the value of parameter q is further increased; but the results indicate that better costs cannot be obtained whereas the algorithm gets much slower. Therefore, to keep a balance for ALNS speed and accuracy we adopt ALNS-15-30 (q' is randomly selected in 15–30%) as our default setting in the following tests.

5.3. Testing on benchmarks

To further assess the performance of ALNS, we test the algorithm on benchmarks from the literature and compare our results with solutions provided by existing algorithms. As stated above, Bredström and Rönnqvist (2008) study the VRPTW with temporal precedence and synchronization constraints. They generated 30 benchmark instances to simulate the HHC staff scheduling

Table 4
Computational results on 200-customer test instances.

Instance	ALNS-5-15			ALNS-5-30			ALNS-15-30			ALNS-15-50			ALNS-30-50		
	Best	*Avg*	*Time(s)*	*Best*	*Avg*	*Time(s)*	*Best*	*Avg*	*Time(s)*	*Best*	*Avg*	*Time(s)*	*Best*	*Avg*	*Time(s)*
C1_2_6_10	3167.48	3172.13	504.9	**3167.46**	3168.43	873.1	**3167.46**	3167.5	1123.5	**3167.46**	3167.5	1630.4	**3167.46**	3167.47	2075.3
C1_2_6_20	3544.63	3546.96	1074.9	3542.51	3544.09	1658.8	**3542.23**	3543.32	2037.3	3542.71	3543.79	3053.7	3542.69	3544.54	3729.4
C1_2_6_40	3957.39	3971.8	2281.7	3960.16	3970.44	3533.4	**3957.2**	3968.52	4564.2	3959.26	3974.5	6735.3	**3957.2**	3971.43	8156.1
C2_2_3_10	2087.55	2104.63	300.4	2068.23	2082.05	550.7	**2047.15**	2087.8	782.4	2056.48	2078.97	1195.1	2065.08	2080.34	1491.0
C2_2_3_20	2322.6	2356.51	392.9	2251.68	2296.8	712.7	2239.99	2253.44	1027.7	**2237.17**	2265.42	1626.9	2251.08	2304.74	2098.1
C2_2_3_40	2887	3022.05	930.2	2961.25	3034.49	1233.9	2876.56	3013.1	1569.7	**2854.84**	2989.1	1942.0	2872.87	3009.91	2721.2
R1_2_9_10	3968.63	3990.19	338.0	3944.52	3959.73	775.5	3944.65	3958.74	984.6	3949.44	3960.79	1520.1	**3940.07**	3955.54	1772.1
R1_2_9_20	4181.8	4218.16	780.8	4173.09	4195.71	1298.6	**4152.7**	4198.26	1725.4	4177.44	4210.23	2602.8	4197.15	4217.32	3089.2
R1_2_9_40	4639.5	4681.37	1940.9	4637.19	4656.32	2836.1	**4631.68**	4650.66	3823.9	4639.21	4654.22	5537.0	4635.05	4656.27	6926.8
R2_2_5_10	3433.21	3474.3	288.7	3393.17	3426.94	597.3	3391.12	3421.49	906.4	**3390.36**	3414.58	1387.5	3390.79	3413.4	1752.9
R2_2_5_20	3637.79	3661.41	499.1	3628.95	3653.06	977.4	**3592.17**	3635.13	1334.2	3621.81	3655.01	2118.1	3594.54	3641.7	2634.6
R2_2_5_40	4305.23	4457.74	1030.1	4421.48	4517	1490.0	4295.69	4555.12	1936.4	4356.93	4486.74	2513.0	**4285.16**	4457.74	3717.6
RC1_2_4_10	2384.89	2404.8	239.5	2378.27	2403.59	472.9	2370.01	2390.46	659.5	2369.11	2390.33	994.9	**2368.97**	2392.05	1324.0
RC1_2_4_20	2697.07	2762.81	394.4	2695.03	2739.8	656.2	2714.35	2748.56	919.2	2716.81	2738.41	1401.7	**2693.95**	2755.93	1830.5
RC1_2_4_40	3355.48	3412.99	739.5	3279.15	3342.1	1023.9	**3197.18**	3290.87	1379.1	3213.91	3329.6	1933.0	3227.7	3327.31	2737.9
RC2_2_8_10	2471.87	2514.71	220.9	2487.46	2508.23	450.6	2465.58	2484.42	606.3	2473.12	2497.62	1023.5	**2449.18**	2488.67	1269.1
RC2_2_8_20	2813.21	2854.25	283.4	2820.05	2855.86	595.3	2795.14	2839.45	827.4	**2776.80**	2842.36	1397.4	2793.62	2834.33	1799.4
RC2_2_8_40	3476.6	3575.96	594.6	3438.65	3528.2	897.2	3371.82	3486.45	1108.8	**3434.77**	3540.22	1672.1	3349.89	3502.26	2481.4
Avg	3296.22	3343.49	713.1	3291.57	3326.82	1146.3	3264.04	3316.29	1517.5	3274.31	3318.86	2238.0	3265.69	3317.83	2867.1

problem, containing from 20–80 customers. In these instances, the customer locations were uniformly distributed throughout a square area with the depot (HHC company) located in the centre. On average, 10% of the customers need simultaneous service from two vehicles. Bredström and Rönnqvist (2008) first solve such test instances with three different objectives, using Cplex solver and their heuristic approach. Bredström and Rönnqvist (2007) present two versions of branch-and-price algorithms to solve these instances. Recently, Afifi et al. (2016) present a simulated annealing based algorithm for this problem. Note that the three objectives proposed by Bredström and Rönnqvist (2008) and adopted by Afifi et al. (2016) are different from our objective expression (1). The first objective of Bredström and Rönnqvist is to minimize the total travel times of all vehicles. Clearly, our original objective expression (1) can cover this objective as a special case if we let $\zeta_k = 0$ for each $k \in K$. The second objective of Bredström and Rönnqvist is to minimize the sum of assigned negative preferences. Bredström and Rönnqvist (2008), and Afifi et al. (2016) introduce a r_{ik} variable to express a negative preference of assigning vehicle k to customer i. For a solution s, the expression $c'(s) = \sum_{k\in K}\sum_{i\in N'}\sum_{j\in V'} r_{ik}x_{ijk}$ is used as this second objective. To create a clear comparison with the results of Bredström and Rönnqvist (2008), and Afifi et al. (2016) on this optimization objective, we modify ALNS algorithm described earlier, i.e., during the ALNS iteration search each solutions s is evaluated as

$$f\prime(s) = c\prime(s) + \alpha P_1(s) + \beta P_2(s). \tag{14}$$

This solution evaluation is adopted in the initial procedure and in the evaluation for each insertion and removal operator. The third objective of Bredström and Rönnqvist is to minimize differences in the attribution of the workload, i.e., balance the workloads among vehicles, which is defined as,

$$c''(s) = Max\left(\left|\sum_{i\in N\prime}\sum_{j\in V\prime}\tau_i x_{ijk} - \sum_{i\in N\prime}\sum_{j\in V\prime}\tau_i x_{ijl}\right|\right), \quad \forall k \in K, l \in K\backslash\{k\}. \tag{15}$$

Similarly, for comparison with the algorithm of previous studies on this objective, we adopt the following criterion in the ALNS algorithm,

$$f''(s) = c''(s) + \alpha P_1(s) + \beta P_2(s). \tag{16}$$

The results of our ALNS algorithm and all previous methods are detailed in Tables 5–7. The columns "*Instance*" and "*N*" provide the instance label and the number of customers. Columns "*MIP*" and "*H*" show the results of the Cplex solver and heuristic by Bredström and Rönnqvist (2008), where sub-columns with the headers "*Cost*" and "*Time*" report the best solution cost found by each method and the computational time in seconds. Column "*BP*" lists the results of the branch-and-price algorithms presented in Bredström and Rönnqvist (2007). They test two variants of their algorithm when considering travel time; those results are denoted separately by columns "BP_1" and "BP_2" in Table 5. Column "*SA*" indicates the results of the simulated annealing algorithm in Afifi et al. (2016). Finally, the last column shows the ALNS solution cost for each instance, including the best and average solution costs of ALNS and average time for a single run. All run times are given in seconds. Note that to establish an accurate and fair comparison with previous studies, Afifi et al. (2016) re-implemented other previous approaches using a more recent version of Cplex 12.6 and an improved computation environment (Intel Xeon 2.67 GHz under Linux). Thus, we adopted the data published by Afifi et al. (2016) directly. In these tables, for each test instance the best-known solutions (BKS) are in bold text, and the last row gives the average of one column.

The results in Tables 5–7 indicate that the performance of ALNS algorithm is satisfied. First, in terms of the first objective (minimizing the total travel time), the ALNS outperforms the methods of Bredström and Rönnqvist (2008), and is competitive with the algorithm of Afifi et al. (2016) in terms of both algorithmic accuracy and speed. Out of the total 30 test instances the ALNS is able to find 24 existing BKS. For three test instances (9 L, 10S and 10 L), ALNS finds new BKS. Although for the 3 remaining instances, SA approach of Afifi et al. outperforms ALNS, the average value of the best solution cost found by ALNS (6.280) is still smaller than the mean value of best known solutions found by SA (6.296). Regarding computational time, the average run time of the proposed ALNS is approximately 4.6 s, whereas the average run time required by the SA of Afifi et al. (2016) is more than 31 s.

Considering the second and third test objectives, the proposed ALNS outperforms the existing approaches. For example, as shown in Table 6, with the objective of minimizing the negative preferences, ALNS is able to find the same or better BKS for each test instance. Moreover, ALNS is able to find new BKS for 5 test instances (9S, 9 M, 9 L, 10 M, and 10 L). In terms of the last objective, balancing the works among vehicles, as shown in Table 7, ALNS dramatically outperforms existing approaches. Out of the 30 total test instances the ALNS is able to find 9 existing BKS and 21 new

Table 5
Computational results on benchmark instances for the total travel time.

			MIP		*H*		*BP_1*		*BP_2*		*SA*		*ALNS*		
Instance	*N*	*N_2*	*Cost*	*Time*	*Best*	*Time*	*Best*	*Time*	*Best*	*Time*	*Best*	*Time*	*Best*	*Avg*	*Time*
1S	20	2	**3.55**	3.43	**3.55**	120.03	**3.55**	1.96	**3.55**	1.12	**3.55**	0.02	**3.55**	3.55	0.06
1M	20	2	**3.55**	14.48	**3.55**	120.25	**3.55**	221.93	**3.55**	3.69	**3.55**	0.02	**3.55**	3.55	0.06
1L	20	2	**3.39**	76.71	**3.39**	120.48	**3.39**	107.41	**3.39**	11.91	**3.39**	0.03	**3.39**	3.39	0.06
2S	20	2	**4.27**	0.22	**4.27**	120.07	**4.27**	3.28	**4.27**	0.56	**4.27**	0.02	**4.27**	4.27	0.08
2M	20	2	**3.58**	25.97	**3.58**	120.11	**3.58**	8.12	**3.58**	3.2	**3.58**	0.03	**3.58**	3.58	0.09
2L	20	2	**3.42**	183.11	**3.42**	120.95	**3.42**	2.72	**3.42**	7.41	**3.42**	0.03	**3.42**	3.42	0.09
3S	20	2	**3.63**	1.79	**3.63**	120.26	**3.63**	14.17	**3.63**	3.84	**3.63**	0.02	**3.63**	3.63	0.09
3M	20	2	**3.33**	21.24	**3.33**	120.19	**3.33**	17.57	**3.33**	4.31	**3.33**	0.03	**3.33**	3.33	0.09
3L	20	2	**3.29**	96.47	**3.29**	120.6	**3.29**	42.78	**3.29**	1.44	**3.29**	0.02	**3.29**	3.29	0.09
4S	20	2	**6.14**	30.9	**6.14**	120.16	**6.14**	14.02	**6.14**	1.54	**6.14**	0.02	**6.14**	6.14	0.08
4M	20	2	**5.67**	1380	**5.67**	120.15	**5.67**	27.53	**5.67**	2.55	**5.67**	0.05	**5.67**	5.67	0.09
4L	20	2	**5.13**	3600	**5.3**	120.04	**5.13**	9.74	**5.13**	7.69	**5.13**	0.09	**5.13**	5.13	0.09
5S	20	2	**3.93**	6.99	**3.93**	120.13	**3.93**	2.84	**3.93**	2.9	**3.93**	0.03	**3.93**	3.93	0.09
5M	20	2	**3.53**	6.2	**3.53**	120.09	**3.53**	57.04	**3.53**	9.1	**3.53**	0.03	**3.53**	3.53	0.08
5L	20	2	**3.34**	225.23	**3.34**	120.85	**3.34**	9.11	**3.34**	5.15	**3.34**	0.03	**3.34**	3.34	0.09
6S	50	5	**8.14**	3600	**8.14**	600.94	**8.14**	3600	**8.14**	197	**8.14**	13.97	**8.14**	8.14	1.67
6M	50	5	**8.14**	3600	**11.63**	609.58	**7.71**	3600	**7.7**	3600	**7.7**	26.68	**7.70**	7.70	1.63
6L	50	5	–	3600	–	624.06	**7.14**	3279	**7.14**	3600	**7.14**	15.86	**7.14**	7.14	1.47
7S	50	5	–	3600	8.97	603.97	**8.39**	14.72	**8.39**	169	**8.39**	15.08	**8.39**	8.39	1.71
7M	50	5	12.66	3600	–	648.02	7.67	3600	**7.48**	3600	**7.48**	18.34	**7.48**	7.52	1.94
7L	50	5	12.66	3600	–	645.33	**6.88**	3600	**6.88**	3600	**6.88**	15.92	**6.88**	6.89	1.66
8S	50	5	–	3600	–	657.03	**9.54**	931	**9.54**	850	**9.54**	25.13	**9.54**	9.59	2.11
8M	50	5	–	3600	–	632.61	**8.54**	3600	**8.54**	3490	**8.54**	15.01	**8.54**	8.56	1.85
8L	50	5	–	3600	–	618.63	8.62	3600	8.11	3600	**8.0**	24.51	8.02	8.03	1.92
9S	80	8	–	3600	–	626.26	–	3600	12.21	3600	**11.93**	150.52	11.95	12.06	22.50
9M	80	8	–	3600	–	612.19	11.74	3600	11.04	3600	**10.92**	292.17	**10.92**	11.02	23.03
9L	80	8	–	3600	–	607.36	11.11	3600	10.89	3600	10.49	207.17	**10.43**	10.59	21.54
10S	80	8	–	3600	–	604.46	–	3600	9.13	3600	8.6	16.1	**8.54**	8.57	20.10
10M	80	8	–	3600	–	705.2	8.54	3600	8.1	3600	**7.62**	52.75	7.63	7.65	16.20
10L	80	8	–	3600	–	631.39	–	3600	–	3600	7.75	51.89	**7.36**	7.39	17.00
Avg											6.296	31.39	6.280	6.30	4.59

Table 6
Computational results on benchmark instances for the sum of negative preferences.

			MIP		*H*		*BP*		*SA*		*ALNS*		
Instance	*N*	*N_2*	*Cost*	*Time*	*Best*	*Time*	*Best*	*Time*	*Best*	*Time*	*Best*	*Avg*	*Time*
1S	20	2	**−114.03**	1.05	**−114.03**	3600	**−114.03**	1.27	**−114.03**	0.03	**−114.03**	−114.03	0.08
1M	20	2	**−117.8**	1.04	**−117.8**	3600	**−117.8**	1.68	**−117.8**	0.02	**−117.8**	−117.80	0.07
1L	20	2	**−118.51**	0.52	−117.8	3600	**−118.51**	2.55	**−118.51**	0.04	**−118.51**	−118.51	0.08
2S	20	2	**−92.09**	0.58	**−92.09**	3600	**−92.09**	0.6	**−92.09**	0.05	**−92.09**	−92.09	0.06
2M	20	2	**−104.81**	32.94	**−104.81**	3600	**−104.81**	2.3	**−104.81**	0.04	**−104.81**	−104.81	0.07
2L	20	2	**−107.64**	427.74	**−107.64**	3600	**−107.64**	6.44	**−107.64**	0.38	**−107.64**	−107.22	0.07
3S	20	2	**−99.49**	0.95	**−99.49**	3600	**−99.49**	1.66	**−99.49**	0.02	**−99.49**	−99.49	0.07
3M	20	2	**−106.59**	2.79	**−106.59**	3600	**−106.59**	2.01	**−106.59**	0.07	**−106.59**	−106.59	0.07
3L	20	2	**−107.87**	1.95	**−107.87**	3600	**−107.87**	2.63	**−107.87**	0.14	**−107.87**	−107.87	0.07
4S	20	2	**−100**	2.22	**−100**	3600	**−100**	1.72	**−100**	0.03	**−100**	−100.00	0.08
4M	20	2	**−106.72**	68.26	**−106.72**	3600	**−106.72**	2.36	**−106.72**	0.07	**−106.72**	−106.72	0.09
4L	20	2	**−109.27**	170.25	**−109.27**	3600	**−109.27**	5.04	**−109.27**	0.13	**−109.27**	−109.27	0.09
5S	20	2	**−76.29**	0.26	**−76.29**	3600	**−76.29**	0.64	**−76.29**	0.02	**−76.29**	−76.29	0.08
5M	20	2	**−76.29**	1.08	**−76.29**	3600	**−76.29**	1.28	**−76.29**	0.03	**−76.29**	−76.29	0.08
5L	20	2	**−84.21**	16.33	**−84.21**	3600	**−84.21**	2.21	**−84.21**	0.04	**−84.21**	−84.21	0.08
6S	50	5	**−370.06**	1452.76	**−370.06**	3600	**−370.06**	150.63	**−370.06**	0.7	**−370.06**	−369.55	0.90
6M	50	5	−372.4	3600	−374.257	3600	**−379.88**	247.88	**−379.88**	25.26	**−379.88**	−376.97	0.72
6L	50	5	–	3600	−368.876	3600	**−387.2**	474.15	**−387.2**	16.33	**−387.20**	−385.74	0.73
7S	50	5	–	3600	−296.725	3600	**−401.11**	291.29	**−401.11**	0.58	**−401.11**	−401.11	0.75
7M	50	5	–	3600	−368.565	3600	**−406.17**	86.7	**−406.17**	6.48	**−406.17**	−406.17	0.66
7L	50	5	–	3600	−355.716	3600	**−407.48**	710.62	**−407.48**	2.53	**−407.48**	−407.48	0.57
8S	50	5	–	3600	–	3600	**−380.76**	135.39	**−380.76**	26.12	**−380.76**	−372.16	0.96
8M	50	5	–	3600	–	3600	**−403.57**	290.77	**−403.57**	59.34	**−403.57**	−401.98	0.85
8L	50	5	–	3600	–	3600	**−407.48**	362.18	**−407.48**	20.51	**−407.48**	−406.79	0.80
9S	80	8	–	3600	–	3600	−552.65	3600	−581.12	117.4	**−619.71**	−611.70	9.01
9M	80	8	–	3600	–	3600	−463.82	3600	−656.5	10.9	**−660.55**	−655.33	6.82
9L	80	8	–	3600	–	3600	−663.47	3600	−666.5	17.81	**−669.73**	−665.98	6.73
10S	80	8	–	3600	–	3600	**−675.81**	3600	**−675.81**	162.42	**−675.81**	−673.70	5.30
10M	80	8	–	3600	–	3600	−685.31	3600	−686.75	150.63	**−686.77**	−684.46	4.47
10L	80	8	–	3600	−445.027	3600	−691.34	3600	−691.48	270.26	**−691.86**	−690.25	4.06
Avg							−293.257	812.80	−300.783	29.61	−302.325	−301.02	1.78

Table 7
Computational results on benchmark instances for the fairness objective.

			MIP		*H*		*SA*		*ALNS*		
Instance	*N*	N_2	*Cost*	*Time*	*Best*	*Time*	*Best*	*Time*	*Best*	*Avg*	*Time*
1S	20	2	**0**	444.1	0.03	3600	**0**	0.3	**0**	0.03	0.07
1M	20	2	**0**	0.14	**0**	3600	**0**	0.45	**0**	0.03	0.07
1L	20	2	**0**	0.16	**0**	3600	**0**	0.61	**0**	0.03	0.07
2S	20	2	0.04	3600	**0.01**	3600	**0.01**	0.3	**0.01**	0.03	0.07
2M	20	2	0.04	3600	**0.01**	3600	**0.01**	0.6	**0.01**	0.04	0.08
2L	20	2	–	3600	**0.01**	3600	**0.01**	0.81	**0.01**	0.03	0.08
3S	20	2	0.06	3600	**0.01**	3600	**0.01**	0.49	**0.01**	0.03	0.07
3M	20	2	0.06	3600	**0.01**	3600	**0.01**	0.65	**0.01**	0.03	0.07
3L	20	2	0.06	3600	**0.01**	3600	**0.01**	0.7	**0.01**	0.02	0.07
4S	20	2	0.13	3600	0.07	3600	0.07	0.61	**0.06**	0.06	0.07
4M	20	2	0.08	3600	0.03	3600	0.02	0.73	**0.02**	0.04	0.07
4L	20	2	0.08	3600	0.02	3600	0.02	0.46	**0.02**	0.03	0.07
5S	20	2	0.08	3600	0.01	3600	0.01	0.39	**0.01**	0.04	0.07
5M	20	2	0.08	3600	0.01	3600	0.01	0.69	**0.01**	0.03	0.07
5L	20	2	0.06	3600	0.03	3600	0.01	1.13	**0.01**	0.02	0.07
6S	50	5	–	3600	0.7	3600	0.11	2.35	**0.05**	0.07	0.76
6M	50	5	–	3600	0.56	3600	0.07	18.32	**0.07**	0.13	0.72
6L	50	5	–	3600	1.75	3600	0.12	0.57	**0.08**	0.14	0.73
7S	50	5	–	3600	1.59	3600	0.18	1.04	**0.06**	0.08	0.80
7M	50	5	–	3600	–	3600	0.15	0.97	**0.07**	0.09	0.72
7L	50	5	0.77	3600	–	3600	0.14	0.84	**0.06**	0.15	0.70
8S	50	5	–	3600	–	3600	0.36	1.36	**0.10**	0.14	1.00
8M	50	5	–	3600	–	3600	0.33	0.84	**0.08**	0.12	0.92
8L	50	5	–	3600	–	3600	0.32	1.19	**0.10**	0.17	0.84
9S	80	8	–	3600	–	3600	0.45	4.09	**0.13**	0.22	9.57
9M	80	8	–	3600	–	3600	0.23	3.43	**0.09**	0.19	7.97
9L	80	8	–	3600	–	3600	0.46	3.35	**0.14**	0.20	7.21
10S	80	8	–	3600	–	3600	0.22	3.57	**0.06**	0.10	6.37
10M	80	8	–	3600	–	3600	0.28	5.2	**0.07**	0.11	5.32
10L	80	8	–	3600	–	3600	0.25	6.22	**0.11**	0.19	5.09
Avg							0.129	2.08	0.05	0.086	1.66

Table 8
ALNS and SA results on 100-customer test instances.

Instance	BKS	ALNS			SA		Gap (%)
		Best	Avg	Time(s)	Best	Avg	
C106_100_5	**1039.96**	**1039.96**	1039.96	49.5	**1039.96**	1040.68	0.00
C106_100_10	**1146.47**	**1146.47**	1146.47	46.9	**1146.47**	1146.86	0.00
C106_100_20	**1310.21**	**1310.21**	1315.14	40.8	1311.28	1317.65	−0.08
C201_100_5	**773.11**	**773.11**	773.11	41.8	**773.11**	773.17	0.00
C201_100_10	**850.81**	**850.81**	850.81	38.2	**850.81**	850.81	0.00
C201_100_20	**940.99**	**940.99**	940.99	42.1	**940.99**	940.99	0.00
R110_100_5	**1161.42**	**1161.42**	1170.72	77.1	**1161.42**	1178.37	0.00
R110_100_10	**1222.08**	**1222.08**	1230.26	70.9	**1222.08**	1229.76	0.00
R110_100_20	**1372.26**	1398.93	1407.72	63.7	1397.44	1411.17	0.11
R209_100_5	**937.40**	**937.40**	951.64	45.0	939.60	953.33	−0.23
R209_100_10	**1003.72**	1004.29	1013.39	75.4	**1003.72**	1015.18	0.06
R209_100_20	**1157.87**	**1157.87**	1179.76	56.0	**1157.87**	1179.98	0.00
RC101_100_5	**1762.22**	1765.06	1765.59	155.6	1763.63	1767.54	0.08
RC101_100_10	**1891.49**	**1891.49**	1905.44	96.7	1899.11	1917.48	−0.40
RC101_100_20	**2096.26**	**2096.26**	2106.84	107.9	2103.62	2117.28	−0.35
RC205_100_5	**1301.23**	**1301.23**	1315.03	61.3	1306.74	1320.30	−0.42
RC205_100_10	**1376.32**	**1376.32**	1387.93	162.4	1378.27	1395.89	−0.14
RC205_100_20	**1571.22**	**1571.22**	1602.68	77.1	1595.57	1615.20	−1.55
Avg	1273.06	1274.73	1283.53	72.7	1277.32	1287.31	−0.16

BKS. For relatively large and difficult test instances, e.g., 8S-10 L, the best-solution cost of ALNS is only half or even less than one third of the best-solution costs of the previous BKS. The average of the best solution costs found by the SA of Afifi et al. (2016) is 0.129, whereas the mean value of best solutions of ALNS is only 0.05. In conclusion, we can state that our approach outperforms previous approaches on the existing benchmark test instances.

5.4. Testing on new lager benchmark instances

As stated above, the available benchmark instances generated by Bredström and Rönnqvist (2008) only contains 20–80 customers. To extensively examine the proposed ALNS algorithm, we compare it with existing VRPTWSyn best algorithm, the simulated annealing algorithm of Afifi et al. (2016), on new larger-scale test instances of Section 5.1, in which each instance has 100 or 200 customers. As our demands to get the code of simulated annealing have been unfruitful, and for another hand to establish an accurate comparison, we develop our own implementation. All the experimental results are reported based on the same hardware and software environments. The only modification to simulated annealing algorithm of Afifi et al. (2016) is stop criterion: for each test instance it stops when its running time equals the corresponding computation time of ALNS. All the computational results

Table 9
ALNS and SA results on 200-customer test instances.

Instance	BKS	ALNS			SA		Gap (%)
		Best	Avg	Time(s)	Best	Avg	
C1-2-6-10	**3167.46**	**3167.46**	3167.5	1123.5	3167.48	3171.57	0.00
C2-2-3-10	**3542.23**	**3542.23**	3543.32	2037.3	3542.88	3545.53	−0.02
R1-2-9-10	**3957.20**	**3957.20**	3968.52	4564.2	**3957.20**	3968.77	0.00
R2-2-5-10	**2047.15**	**2047.15**	2087.8	782.4	2071.51	2104.07	−1.19
RC1-2-4-10	**2237.17**	2239.99	2253.44	1027.7	2267.22	2333.10	−1.22
RC2-2-8-10	**2854.84**	2876.56	3013.1	1569.7	2887.00	3016.72	−0.36
C1-2-6-20	**3940.07**	3944.65	3958.74	984.6	3944.52	3977.08	0.00
C2-2-3-20	**4152.70**	**4152.70**	4198.26	1725.4	4173.09	4202.85	−0.49
R1-2-9-20	**4631.68**	**4631.68**	4650.66	3823.9	4639.50	4671.00	−0.17
R2-2-5-20	**3390.36**	3391.12	3421.49	906.4	3407.56	3445.83	−0.48
RC1-2-4-20	**3592.17**	**3592.17**	3635.13	1334.2	**3592.17**	3649.93	0.00
RC2-2-8-20	**4285.16**	4295.69	4555.12	1936.4	4305.23	4456.47	−0.22
C1-2-6-40	**2368.97**	2370.01	2390.46	659.5	2380.30	2402.54	−0.43
C2-2-3-40	**2693.95**	2714.35	2748.56	919.2	2897.07	2945.32	−6.73
R1-2-9-40	**3197.18**	**3197.18**	3290.87	1379.1	3288.79	3382.97	−2.87
R2-2-5-40	**2449.18**	2465.58	2484.42	606.3	2471.87	2514.24	−0.26
RC1-2-4-40	**2776.80**	2795.14	2839.45	827.4	2820.05	2855.24	−0.89
RC2-2-8-40	**3349.89**	3371.82	3486.45	1108.8	3449.37	3541.80	−2.30
Avg	3257.45	3264.04	3316.29	1517.6	3292.38	3343.61	−0.98

obtained on these large benchmark instances are summarized in Tables 8 and 9. We present the *best* and *average* solution costs, and mean running time in seconds for both algorithms. Column "*BKS*" provides the best known solution cost for each test instance which is obtained by the experiments for ALNS parameter q setting (in Section 5.2). Column "*Gap*" is the percentage deviation between ALNS and simulated annealing best solution costs, calculated by: $100\times$ (ALNS−SA)/ALNS. For each test instance the best known solution (BKS) is in bold text.

We can assert that the performance of ALNS is better than simulated annealing algorithm for both 100-customer and 200-customer test instances. For 15 out of 18 100-customer test instances, ALNS is able to find the best known solutions, whereas the simulated annealing algorithm finds 9 best known solutions. The average value of best solution costs found by ALNS is slightly better than that of simulated annealing with a deviation of 0.16%. The superiority of ALNS is higher with respect to larger 200-customer test instances. ALNS finds 8 best known solutions, whereas simulated annealing is able to get 2 best known solutions. Among all 18 such 200-customer test instances, ALNS finds better solutions than simulated annealing on 16 instances; in terms of left 2 test instances, ALNS and simulated annealing find the same best solutions. The mean percentage deviation between their best solutions costs increase to about 1%.

6. Conclusions

This paper investigates a special variant of the vehicle routing problem with time windows, the *vehicle routing problem with time windows and synchronized visits*. The problem is of interest because of its theoretical complexity and the importance in many applications such as the daily logistics of home health care companies. We formulate the problem as a mixed integer programming model. Because the synchronization constraints interconnect various routes of the vehicles, solving the proposed problem is more complicated than solving standard vehicle routing problems, especially when using neighborhood search-based methods. We propose an efficient Adaptive Large Neighborhood Search heuristic to solve this problem. We test benchmark instances from the literature with three different objectives. Experimental results show that our solution method outperforms existing solution methods, finding new best-known solutions with less computation time. We also examine the proposed algorithm with existing best heuristic on new generated larger-scale instances. The superiority of ALNS is more obvious with respect to such larger test instances.

Acknowledgments

This work was supported by Research Grant from National Natural Science Foundation of China (71672112, 71501109), and Shanghai Science and Technology Committee Major Program No. 17DZ1101202.

References

Afifi, S., Dang, D.-C., Moukrim, A., 2016. Heuristic solutions for the vehicle routing problem with time windows and synchronized visits. Optim. Lett. 10, 511–525.
Bredström, D., Rönnqvist, M., 2008. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. Eur. J. Oper. Res. 191, 19–31.
Bredstrom, D., Rönnqvist, M., 2007. A Branch and Price Algorithm for the Combined Vehicle Routing and Scheduling Problem with Synchronization Constraints.
Cordeau, J.F., Laporte, G., Mercier, A., 2001. A unified tabu search heuristic for vehicle routing problems with time windows. J. Oper. Res. Soc. 52, 928–936.
Desaulniers, G., Lessard, F., Hadjar, A., 2008. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. Transp. Sci. 42, 387–404.
Dohn, A., Kolind, E., Clausen, J., 2009. The manpower allocation problem with time windows and job-teaming constraints: a branch-and-price approach. Comput. Oper. Res. 36, 1145–1157.
Dohn, A., Rasmussen, M.S., Larsen, J., 2011. The vehicle routing problem with time windows and temporal dependencies. Networks 58, 273–289.
Drexl, M., 2012. Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. Transp. Sci. 46, 297–316.
Eveborn, P., Flisberg, P., Rönnqvist, M., 2006. Laps Care—an operational system for staff planning of home care. Eur. J. Oper. Res. 171, 962–976.
Fikar, C., Hirsch, P., 2015. A matheuristic for routing real-world home service transport systems facilitating walking. J. Clean. Prod. 105, 300–310.
Fikar, C., Juan, A.A., Martinez, E., Hirsch, P., 2016. A discrete-event driven metaheuristic for dynamic home service routing with synchronised trip sharing. Eur. J. Ind. Eng. 10, 323–340.
Gehring, H., Homberger, J., 1999. A Parallel Hybrid Evolutionary Metaheuristic for the Vehicle Routing Problem With Time Windows.
Haddadene, S.R.A., Labadie, N., Prodhon, C., 2014. GRASP for the vehicle routing problem with time windows, synchronization and precedence constraints, wireless and mobile computing, networking and communications (WiMob). In: 2014 IEEE 10th International Conference on. IEEE, pp. 72–76.
Laporte, G., 2016. Scheduling issues in vehicle routing. Ann. Oper. Res. 236, 463–474.
Rabeh, R., Saïd, K., Eric, M., 2011. Collaborative model for planning and scheduling caregivers' activities in homecare. IFAC Proc. Vol. 44, 2877–2882.
Rasmussen, M.S., Justesen, T., Dohn, A., Larsen, J., 2012. The home care crew scheduling problem: preference-based visit clustering and temporal dependencies. Eur. J. Oper. Res. 219, 598–610.
Redjem, R., Marcon, E., 2016. Operations management in the home care services: a heuristic for the caregivers' routing problem. Flex. Serv. Manuf. J. 28, 280–303.
Ropke, S., Pisinger, D., 2006. An Adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp. Sci. 40, 455–472.

Rousseau, L.-M., Gendreau, M., Pesant, G., 2003. The Synchronized Vehicle Dispatching Problem. Centre for Research on Transportation.
Rousseau, L.-M., Gendreau, M., Pesant, G., 2013. The synchronized dynamic vehicle dispatching problem. INFOR: Inf. Syst. Oper. Res. 51, 76–83.
Salazar-Aguilar, M.A., Langevin, A., Laporte, G., 2012. Synchronized arc routing for snow plowing operations. Comput. Oper. Res. 39, 1432–1440.
Solomon, M.M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper. Res. 35, 254–265.
Thomsen, K., 2006. Optimization on Home Care. Technical University of Denmark, Lyngby, Denmark DTU, DK-2800 Kgs.