The Vehicle Rescheduling Problem: Model and Algorithms

Jing-Quan Li and Pitu B. Mirchandani
Department of Systems and Industrial Engineering, The University of Arizona, Tucson, Arizona 85721

Denis Borenstein
Management School, Universidade Federal do Rio Grande do Sul, R. Washington Luis 855, Porto Alegre 90010-460, RS, Brazil

When a vehicle on a scheduled trip breaks down, one or more vehicles need to be rescheduled to serve the customers on that trip with minimum operating and delay costs. The problem of reassigning vehicles in real-time to this *cut trip* as well as to other scheduled trips with given starting and ending times, is referred to as the *vehicle rescheduling problem* (VRSP). This paper considers modeling, algorithmic, and computational aspects of the single-depot VRSP. The paper formulates a model for this problem and develops several fast algorithms to solve it, including parallel synchronous auction algorithms. The concept of the common feasible network (CFN) is introduced to find a good set of initial "prices" for speeding up the auction algorithm. Computational experiments on randomly generated problems are described. Computational results show that, for small problems, all of the developed algorithms demonstrate very good computational performances. For large problems, parallel CFN-based auction algorithms provide the optimal solution with much smaller computation times. © 2007 Wiley Periodicals, Inc. NETWORKS, Vol. 50(3), 211–229 2007

Keywords: vehicle scheduling; rescheduling; auction algorithm; parallel processing

1. INTRODUCTION

The vehicle rescheduling problem (VRSP) arises when a previously assigned trip is disrupted. A traffic accident, a medical emergency and a breakdown of a vehicle are examples of possible disruptions that demand the rescheduling of vehicle trips. The VRSP can be approached as a dynamic version of the classical vehicle scheduling problem (VSP) where assignments are generated dynamically.

The VRSP arises in a wide array of practical applications. Instances of VRSP occur in school bus routing, operational planning of public transportation systems, industrial/hospital refuse collection, mail delivery, telecommunication systems, etc. Because of our initial motivation, which arose from bus rescheduling, in this paper special contextual reference is made to the VRSP problem in public transit systems, where a trip disruption results not only in the increase of operational and delay costs, but also in considerations to accommodate passengers whose plans have been disrupted by the vehicle breakdown.

We note that the VRSP is one of two major subproblems that must be considered in the real-time *operational planning* of a transportation/logistic system; the other is crew rescheduling. Although in strategic planning, the vehicle scheduling and crew scheduling problems are being studied, separately as well as in an integrated fashion [7, 13], the corresponding rescheduling problems have received little attention from a real-time operational planning perspective. Nevertheless, we have decided to focus this paper on solving the VRSP. Integration of crew rescheduling in real-time operations is a topic of future research.

The literature describes several different approaches to solve the VSP [12], but the literature on VRSP is scant. However, when fleet size is limited and disruptions are frequent, good automated rescheduling tools to assist decision makers become important. Very few companies or agencies use automated rescheduling policies, and there is a gap in availability of algorithms for VRSP. The goal of this research is to address this gap. In particular, the single-depot VRSP is modeled, and algorithms that solve this problem in a reasonable amount of time are proposed.

The most pertinent decision for VRSP is which vehicle should backup the disrupted trip. The existence of several alternatives generates, in comparison to VSP, several possible *feasible networks* for the problem, each one corresponding to a possible choice of a *backup vehicle*. The selection of the backup vehicle involves several factors such as the time when the trip was disrupted, the position of the remaining vehicles, the available capacity of the potential backup vehicles, and the itinerary compatibility among trips. The existence of several possible feasible networks

Received November 2004; accepted February 2007
Correspondence to: P. B. Mirchandani; e-mail: pitu@sie.arizona.edu
Contract grant sponsor: CAPES, USDOT/FHWA, Arizona DOT
DOI 10.1002/net.20199
Published online in Wiley InterScience (www.interscience.wiley.com).

makes the VRSP a very interesting but difficult problem to solve.

This paper has the following major objectives: (i) to model the single-depot VRSP (SDVRSP), and (ii) based on previous algorithms developed for the VSP, to describe several possible algorithms for solving the problem, including a *parallel auction algorithm* specifically implemented to solve the SDVRSP. To improve the performance of the algorithms, we develop the notion of the *common feasible network* (CFN). Since the difference among feasible networks is small, one promising approach is to perform the algorithm in two stages. In the first stage, the algorithm is carried out for a reduced network that does not include the disrupted trip or the backup vehicle candidates. Then an assignment close to the original one is obtained, since this reduced network is very similar to the original feasible networks. In the second stage, we include the disrupted trip and backup vehicle candidates to reconstruct the feasible networks for the VRSP problem and apply the auction algorithms again, considering the initial assignments from the first stage. The second stage is then performed for all candidate backup vehicles. CFN often improves the computational performance of the auction algorithms due to better initial assignments.

The major contributions of this paper to the literature are as follows: (i) the definition of the VRSP, dealing with issues such as common itineraries, available capacities, time constraints, and backup vehicle candidates; (ii) the introduction of the CFN approach for reducing the computational time of auction assignment algorithms; and (iii) the implementation of a fast parallel auction algorithm for solving the VRSP, using message-passing to speed up communication among the several processors.

This paper is organized as follows: Section 2 reviews the literature on VRSP and related problems. Terminology and formal description of the problem are provided in Section 3. Section 4 presents our formulation for the SDVRSP, and Section 5 describes some fast auction-based algorithms for solving the SDVRSP. In Section 6, the CFN-based auction algorithm is proposed to further reduce computation times. Section 7 presents computational experiments to evaluate the performance of the algorithms. In the concluding Section 8, a summary of the results is presented and areas of future research are discussed.

2. LITERATURE REVIEW

Automatic recovery from disruptions is a relatively new operational strategy; therefore literature related to the topic is scarce. Currently, most transit companies typically avoid reassigning trips during operational disruptions by simply using a backup vehicle from the depot. Since VRSP is strongly related to VSP, we briefly review the literature on the state-of-the art on modeling and solving the VSP, for which there is a vast literature.

Overviews of algorithms and applications for the single-depot VSP (SDVSP) and some of its extensions can be found in [1, 5, 10, 12, 18]. The SDVSP has been formulated as a linear assignment problem, a transportation problem, a minimum-cost flow problem, a quasi-assignment problem, and a matching problem in the literature. Bokinge and Hasselstrom [6] propose a minimum-cost flow approach that uses a significant reduction on the size of the model in terms of the number of variables at the price of an increased number of constraints. An $O(n^3)$ successive shortest-path algorithm and variations for the SDVSP were proposed in [15, 24, 30]. Paixão and Branco [27] propose an $O(n^3)$ quasi-assignment algorithm that is especially designed for the SDVSP. Freling et al. [18] use a quasi-assignment model and employ a forward/reverse auction algorithm for the solution. Computational results presented by [18] show that this approach significantly outperforms the previous approaches that are based on minimum-cost flow and linear assignment models.

The majority of the research on real-time schedule recovery relates to the dynamic vehicle routing problem (VRP), the airline operations recovery problem and the train rescheduling problem.

The dynamic vehicle routing problem, where new requests arise in midst of operations, has gained increasing attention since the late eighties. Recent surveys on dynamic VRP can be found in [19, 20, 28]. Tabu search [23], genetic algorithm [21], assignment, and insertion-based heuristics [17], approximate dynamic programming [31], and nearest-vehicle based heuristic [16] have been proposed to consider online requests and uncertain travel time in the dynamic VRP problem. Yang et al. [34] have developed a general framework for the dynamic vehicle routing problem in which new requests can arrive during operations.

The literature on airline schedule recovery includes Carlson [8], Lettovský [25], and Teodorović and Stojković [32]. These research efforts develop exact optimization models that reschedule flight legs and reroute aircraft by minimizing rerouting and cancellation costs. These exact algorithms consider only several hundreds of flights. Rosenberger et al. [29] present a heuristic method for selecting which aircraft can be rerouted prior to generating new routes, allowing to solve many large recovery instances quickly. However, when applied to real-life problems, consisting of 469 flights and 96 planes, the method, on average, has required more than 50,000 sec in CPU time to solve recovery instances.

Some research has been reported on the train rescheduling problem. Chiu et al. [11] design a labeling-based heuristic to reduce time table disruptions incurred by accidents or train delays. Medanic and Dorfman [26] propose a discrete-event model and a travel advance strategy to produce an efficient train schedule, and to reschedule in case of disruptions. Walker et al. [33] utilize a branch-and-bound method to minimize the deviation of the new schedule obtained from the initial schedule, due to train delays.

To the best of our knowledge, the only contribution towards solving the dynamic VSP is due to Huisman et al. [22] who proposed an approach to the problem by solving a sequence of formulated optimization problems. Their work

is motivated to design robust vehicle schedules that avoid trips starting late in environments characterized by significant traffic jams.

Although the literature has interesting and useful ideas towards the development of automated recovery tools, there is a gap in this literature related to the consideration of vehicle breakdown. Since vehicle breakdowns have not been explicitly considered in train rescheduling [11, 26], dynamic VSP [22], or dynamic VRP [16, 17, 21, 23, 31, 34], most of results available are not directly applicable to our problem. Although aircraft breakdown is considered in some studies of airline operations recovery, most of the available algorithms [8, 25, 29, 32] involve a small number of trips and aircrafts. The majority of these algorithms are computationally intensive, requiring thousands of CPU seconds to solve recovering instances involving hundreds of trips and dozens of planes. As a consequence, it is unclear whether they can be directly applied to our more generic VRSP problem, which can involve thousands of trips and hundreds of vehicles.

3. PROBLEM DESCRIPTION

We first introduce some definitions and notation to describe the vehicle rescheduling problem. To relate to a cut in a graph, we refer to a disrupted trip due to a disabled vehicle, or a vehicle that is effectively inoperable, as a *cut trip*. *Breakdown point* is the point in the cut trip where the trip is disrupted. *Current trip* is the trip on which a vehicle is running. It includes both regular and *deadheading* (movement of a vehicle to a destination without serving any passenger) trips. The vehicle that serves the remaining passengers in the cut trip, referred to as the *backup vehicle* in the sequel, can be identified from the trip just served, which may be referred to as the *backup trip*. Then, essentially, the vehicle assignment problem is to assign (the vehicle from) a node representing a backup trip, or assign (a vehicle from) the depot, to the passengers of the cut trip.

Trips i and j are a *compatible pair of trips* if the same vehicle can reach the starting point of trip j after it finishes trip i. A *route* is a sequence of trips in which all consecutive pairs of trips in the sequence are compatible. Trip i is an *itinerary compatible trip* with trip j if trip i shares the same itinerary of cut trip j from the breakdown point until the ending point of trip j.

The SDVRSP can be defined as follows. Given a depot and a series of trips with fixed starting and ending times, given the travel times between all pairs of locations, and given a cut trip, find a feasible minimum-cost reschedule in which (1) each vehicle performs a feasible sequence of trips, and (2) all passengers or cargo (if there are any) on the cut trip are served. Unlike the SDVSP, in which the fixed capital cost is dominant, the SDVRSP problem focuses on operating and delay costs. Furthermore, in order that transit crews can be reassigned on a new schedule, the computation of SDVRSP needs to be carried out as fast as possible.

There are two possible situations in SDVRSP. The first is when the cut trip is a regular one. Unless the disruption is such that it is impossible to reach the breakdown point, the passengers (or cargo, but from this point we will only refer to serving passengers in rescheduling of buses) of the cut trip have to be served. The solution comprises sending a backup vehicle to the breakdown point, and from there completing the cut trip, and serving its passengers. However, since it is very likely that some trips have common itineraries, the passengers can also be served incidentally by the vehicles that cover compatible itineraries after the breakdown point. Consider the following situation: a backup vehicle changes its original schedule and travels towards the breakdown point, but all the passengers from the disabled vehicle have been incidentally picked up by vehicles that cover compatible itineraries with the cut trip. Such a situation needs to be avoided. If the cut trip is a deadheading trip, the solution is to assign a backup vehicle for the starting location of the next trip of the deadheading vehicle. In both cases, it is very likely that the SDVRSP provides new routes (a reassignment) for a subset of the pre-assigned vehicles. Also, we can expect some delays in the cut trip, mainly in the first situation.

From the viewpoint of the cut trip, the remaining trips can be divided into two categories: (1) unfinished trips that have compatible itineraries with the cut trip from the breakdown point, and (2) the remaining unfinished trips. Figure 1 illustrates these two categories where paths $O_i - D_i$ denotes trip i from origin O_i to destination D_i. The breakdown point is point X on trip 1. Trips 2 and 3 are compatible with trip 1 from point X onwards.

Define set A to be the set of unfinished itinerary compatible trips with the cut trip from the point X, ordered by the arrival time from their current position to point X. Define set B to be the remaining unfinished trips (including a trip directly from the depot).

If the backup trip alternatives are from set A, the backup vehicles can pick up the passengers incidentally. Although a reschedule to service the cut trip may not be necessary, it may be necessary to assign a vehicle from set B to cover the unfinished trips originally assigned to the disabled vehicle. If the backup trip alternatives are from set B, backup vehicles need to travel toward the breakdown point for picking up the passengers on the disabled vehicle.

In VSP, a vehicle can be generally assigned from the depot to any trip before its starting time. Nevertheless, assigning a vehicle from the depot to some future trips in the rescheduling

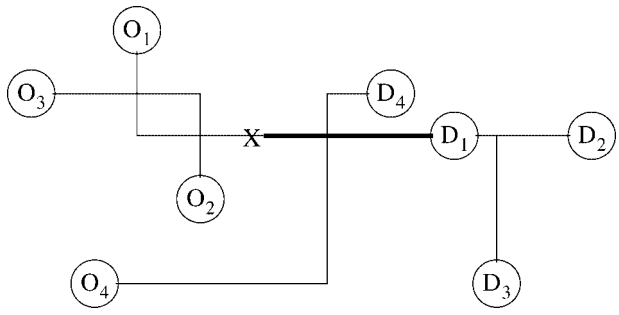

FIG. 1. Example of compatible itinerary trips.

problem may fail if the arrival time of a rescheduled vehicle from the depot to the starting point of a trip is later than the specified starting time of this trip. We may treat the depot s as a special trip (or node) and define its starting time to be breakdown time. This time is used to determine if a backup vehicle from the depot is too late to serve a specified future trip.

In VSP, there is no need to consider assigning a specific vehicle to trips, since all vehicles are identical, and we can assign them arbitrarily after the schedule is determined. However, unlike VSP, VRSP has to take into account this issue, since many vehicles are not at the depot but at different locations in the service region when a vehicle becomes disabled. The corresponding operating costs are also different. This situation creates different possible feasible networks depending on the selected backup vehicle. Whereas there is a unique feasible network in VSP, the VRSP may have several feasible networks (sharing same nodes, but with different arcs connecting them).

Suppose that a regular trip becomes disrupted, and a backup vehicle needs to go to the breakdown point to pick up the passengers. The starting time of this cut trip is dependent on the backup vehicle. The cost and compatible trips are different for alternative backup vehicles, since the serving vehicles are in different positions of the network, rather than at the depot as is assumed in the VSP. This property also provides a solution approach for VRSP; solving VRSP is equivalent to solving a collection of several simple VSPs. It is worth mentioning that, although there may exist many feasible networks, the differences among the networks are the arcs associated with the cut trip and the backup trip candidates.

Vehicle rescheduling needs to be solved quickly, since disruption delays will have a negative influence on system performance, because they simultaneously result in increasing costs and deterioration of the level of service. Moreover, crew members should be notified as quickly as possible of a new schedule. However, it is very difficult to define an upper bound for the time to obtain a new schedule, since this real-time operational issue is highly context dependent. Nevertheless, computational time requirements may be somewhat mitigated by following a strategy where vehicles currently serving regular trips can only change their routes after finishing their current trips. We believe that a reasonable target for the computational time to reschedule should be no more than a minute or two. Therefore, we assume the following in our SDVRSP formulation:

- scheduled trips, except of course the cut trip, cannot suffer delays; and
- there are no restrictions on the number of trips that may be reassigned.

The next few sections describe our model formulation and solution approaches for the SDVRSP. First, in Section 4.1, we present a procedure to define the possible backup trip candidates to construct the set of all feasible networks. This is an essential step to solve the SDVRSP and it is included in all of the developed algorithms. Next, in Section 4.2, an integer linear programming formulation for the SDVRSP is developed.

In order to exploit problem structure, a sequential auction algorithm is described in Sections 5.1 and 5.2, based on the algorithm developed by Freling et al. [18] for the quasi-assignment formulation of the VSP. To further decrease computational time, a parallel version of the auction algorithm that allows the bidding to be simultaneously conducted in several processors is discussed in Section 5.3. Computational time is possibly further reduced when the solution procedure is in two stages, where the first stage provides an initial solution for the second stage; Section 6 describes this procedure.

4. MODELING THE VEHICLE RESCHEDULING PROBLEM

The objective of the SDVRSP is to minimize operating and delay costs over all possible underlying feasible networks. As a consequence, any solution approach needs to explicitly or implicitly generate the set of feasible networks.

4.1. Generation of Feasible Networks

The most important and complicating aspect of the SDVRSP is that the solution is dependent on the current status of vehicles and the available alternatives to serve the cut trip. Each possible configuration of a recovery can be translated as a possible feasible network. These feasible networks share the nodes (the trips), but have different arcs connecting them. The definition of the set of all possible feasible networks is dependent on the pre-assigned configuration of trips, the available capacity of the involved vehicles, and the times to carry out deadheading and regular trips. As commented in Section 3, it is possible to have a different feasible network for each possible backup vehicle candidate. Since vehicles are assigned to trips in the vehicle scheduling problem and network nodes are defined as trips, we can use backup trips (nodes) to define the feasible networks. This subsection describes a procedure to generate the feasible networks based on the available capacities of the involved vehicles, times to service trips in the network and the compatibility of itineraries of the trips.

A capacity problem appears if the backup trip is from set A. It is quite possible that some passengers remain in the disabled vehicle that was servicing the cut trip. If the number of passengers remaining on the cut trip is greater than the vacant capacity of the first vehicle that can serve the backup trip, this vehicle is not enough for picking up all of the passengers. Thus, it is possible that more than one vehicle needs to be sent to the breakdown point. The first vehicle to arrive at the breakdown point picks up some passengers, the next vehicle picks up some more passengers, and so forth until all passengers from the cut trip are served. If the backup vehicle is from set B, it is an empty vehicle. In that case, we assume that one vehicle is enough to pick up all of the passengers from the disabled vehicle.

In addition to the capacity problem, we need to consider time constraints related to the travel times of vehicles on

current trips. It is not possible to select a vehicle (or vehicles) from set A if it has already passed the breakdown point when the disruption occurs. Also, it is important to note that if a vehicle from set B cannot reach the breakdown point earlier than a vehicle from set A that will have enough vacant capacity, then it is inefficient to use a vehicle from the set B to service the cut trip.

To generate the set of possible feasible networks, we need first to determine how many backup trips from set A are sufficient to serve all the passengers from the disabled vehicle. Let $C(i)$ be the empty seats (or remaining capacity) of the backup vehicle from trip i when it reaches the breakdown point. Let $T(i)$ be its arrival time at the breakdown point. The values of both variables may be obtained in real time, based either on information to be provided by the vehicle crew or through an automated GPS-based locator and passenger count system. Otherwise, estimations based on conservative values, provided through historical data, are recommended to assure that passengers on the cut trip will be served.

Let $A(n)$ be the subset of A that includes the first n elements of A. Let P be the number of passengers in the disabled vehicle. Let T_d be the disruption time. Let n^* be the number of backup trips in A that are capable of picking up passengers from the cut trip. n^* satisfies the following properties,

$$\sum_{i \in A(n^*)} C(i) \geq P$$

$$\sum_{i \in A(n^*-1)} C(i) < P$$

$$T(i) \geq T_d, \quad i \in A(n^*).$$

It is possible to define $T(a_{n^*})$ as the time by which the n^* vehicles can serve the passengers on the disabled vehicle, where a_i is the i-th element in set $A(n^*)$. Then, we can determine B^*, the set of backup trips candidates from set B that can pick up the passengers between T_d and $T(a_{n^*})$, by

$$B^* = \{m | T_d \leq T(m) < T(a_{n^*}), \forall m \in B\}.$$

If B^* is empty, then all backup trips are from set $A(n^*)$. In this situation, there is only one feasible network, resulting from eliminating the cut trip from the original network, and the problem can be treated as a VSP. If at least one backup trip candidate is from set B^*, we can construct an arc from this backup trip to the breakdown point to define the corresponding feasible network. We may have several feasible networks since several backup trip candidates may exist.

If there is no value of n^* that respects the above set of inequalities, we set $T(a_{n^*}) \leftarrow \infty$, and set $B^* = B$. In this case, a vehicle from set B has to backup the cut trip.

After determining the set of backup vehicle candidates (and the corresponding backup trips), we can generate the set of all possible feasible networks as follows. Each regular trip is a "node" of the feasible network, graphically represented as a circle with a short interior line segment to indicate starting and ending points of the trip (e.g., Fig. 2). Let b denote the cut trip and K be the set of possible backup trip candidates. "Arcs" in the feasible network correspond to vehicle assignment to trips. For example, the arc from node 2 to node 4 in Figure 2a implies the same vehicle may be assigned to trip 4 after it has served trip 2. Let s and t denote the same depot in the network, where s simply means the depot as a vehicle's starting point, and t as its terminating point. Let $N' = N - \{b\}$ be the set of total remaining trips excluding the cut trip, numbered according to non-decreasing starting times. Let $P \in N$ denote the trips that vehicles are currently serving. If trip i, $i \in P$ is a deadheading trip, its starting time and ending time are set as the current time, since the vehicle on this deadheading trip can be rescheduled right away. Define arc-set $E(k) = E \cup \{(k, b)\}$, where $E = \{(i,j) \in \{N \cup s\} \times N' | [i < j] \wedge [i \text{ and } j \text{ are compatible trips}]\}$ is the set of arcs that correspond to the deadheading trips. A feasible network for backup trip k can be defined as $G(k) = \{V, X(k)\}$ with nodes $V = N \cup \{s, t\}$ and arcs $X(k) = E(k) \cup (s \times P) \cup (N \times t)$, for $k \in K$, where k is the backup trip. Since the trip in P is currently being served by a vehicle, there is no need to allocate another vehicle to cover it. The arcs, $(s \times P)$, are included only for modeling convenience. We define $G = \{G(k) | k \in K\}$ as the set of all feasible networks.

We illustrate with an example, a set of feasible networks and our procedure to construct them. Suppose we need to perform four trips with the starting and ending times as indicated in Table 1, and the travel time from the ending point of

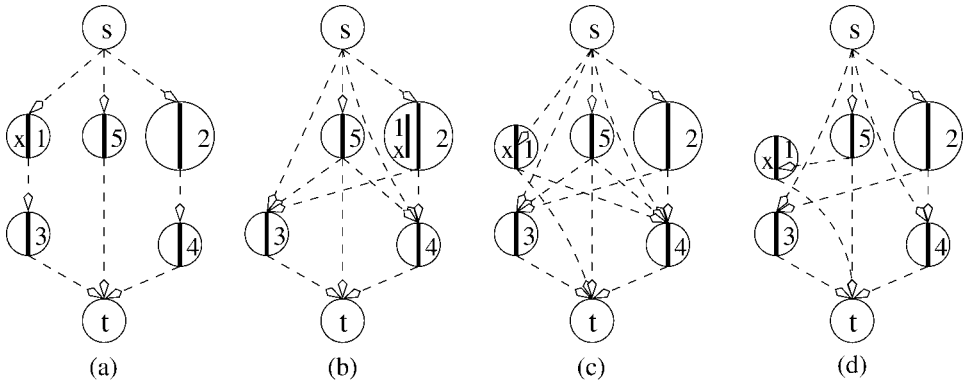

FIG. 2. Examples of feasible networks.

TABLE 1. Travel times.

Trip	Starting time	Ending time	Duration
1	8	14	6
2	1	16	15
3	18	25	7
4	20	28	8
5	3	11	8

each trip (or depot) to the starting point of another trip is a constant (4 time units).

Suppose a vehicle breaks down on trip 1 at a point X at time 11, and the travel time from point X to the ending point of trip 1 is 3 units. Other data for this example are as follows:

- the disabled vehicle is carrying 11 passengers at point X;
- on average, all vehicles have more than 16 available seats;
- trip 2 is an itinerary compatible trip with trip 1 from breakdown point X, and vehicle serving trip 2 has not yet passed the point X;
- the time required from any vehicle to reach the breakdown point after completing its current regular trip is a constant, 3 time units; and
- the time of a vehicle from the depot to the breakdown point is 12 time units.

Thus, set $A = \{2\}$ and set $B = \{0, 3, 4, 5\}$, where the element 0 denotes the assignment of a vehicle from the depot. Since the expected vacant capacity of the vehicle on trip 2 is 16, this vehicle can pick up all passengers. If the vehicles from set B reach point X later than the time when the vehicle on trip 2 arrives at point X, they cannot be considered as backup trip candidates. Times vehicles reach point X from set B are as follows: for trip 3, $25 + 3 = 28$ time units, for trip 4, $28 + 3 = 31$ time units, and for trip 5, $11 + 3 = 14$ time units.

The following cases are described in Figure 2 to illustrate the generation of the possible feasible networks, where Figure 2a shows the initial schedule.

CASE 1 (Suppose vehicle on trip 2 can reach point X at time unit 11). In this case, the only backup trip candidate is trip 2. Although we do not need any backup vehicle to go to the *breakdown point*, it is possible that an extra vehicle is needed to cover some remaining trips assigned to the disabled vehicle. In this case, there is only one feasible network (see Fig. 2b). Trip 2 is finished on time. The feasible network can be constructed by removing trip 1 and the associated arcs.

CASE 2 (Suppose vehicle on trip 2 can reach point X at time unit 13). In this case, the backup vehicle candidates are (i) the vehicle assigned to trip 2, and (ii) an extra vehicle from the depot. If the backup vehicle is the vehicle on trip 2, the feasible network is given in Figure 2b. Figure 2c represents the feasible network if the backup vehicle is an extra vehicle from the depot. The time for this vehicle to finish trip 1 would be $12 + 3 = 15$. If this vehicle was also assigned to trip 3, the time to the starting point of trip 3 is $15 + 4 = 19$, which is more than the specified starting time of trip 3. Therefore trips 1 and 3 become incompatible ($19 > 18$) and this new vehicle cannot be assigned to trip 3.

CASE 3 (Suppose vehicle on trip 2 can reach point X at time unit 15). In this case, we have three possible backup trip candidates, namely trip 5 and the same two as in case 2. If the backup vehicle is on trip 2 or from the depot, the possible feasible networks are depicted in Figures 2b and c, respectively. If the backup vehicle is the vehicle serving trip 5, its time to finish trip 1 is $14 + 3 = 17$; then the time to reach the starting points of trips 3 and 4 is $17 + 4 = 21$. Hence, trips 1 and 3, and trips 1 and 4, become incompatible ($21 > 18$ and $21 > 20$, respectively). The feasible network for this situation is given in Figure 2d. Notice that this case presents three different feasible networks and the solution of the VRSP involves finding the minimum operation and delay cost over the three feasible networks.

Based on the underlying feasible networks, we can model the VRSP as a VSP in each feasible network, and the VRSP optimal schedule is the one with the minimum total cost over all possible *feasible networks*. In order to decrease the number of possible feasible networks if there are a large number, we may define a time limit by which a vehicle has to arrive at the breakdown point. If there are very many elements in B^*, some candidates that exceed this time limit can be deleted using this constraint and, hence, fewer feasible networks will be considered.

4.2. Mathematical Formulation

The SDVRSP can be modeled as a minimization problem over several SDVSP problems, each corresponding to a possible feasible network. Let y_{ij} be a binary decision variable, with $y_{ij} = 1$ if a vehicle is assigned to trip j directly after trip i, $y_{ij} = 0$ otherwise. Let c_{ij} be the vehicle cost on arc $(i,j) \in X(k)$, which is a function of travel time and idle time. Let D_k be the delay cost corresponding to the solution with trip k as the backup trip. The quasi-assignment based formulation for the SDVRSP is as follows:

$$\min_G \left\{ \min \sum_{(i,j)\in X(k)} c_{ij} y_{ij} + D_k \right\}$$

$$st$$

$$\sum_{j:(i,j)\in X(k)} y_{ij} = 1 \quad \forall i \in N$$

$$\sum_{i:(i,j)\in X(k)} y_{ij} = 1 \quad \forall j \in N$$

$$y_{ij} \in \{0,1\} \quad \forall (i,j) \in X(k).$$

The objective of this formulation is to find a schedule with the minimal operating plus delay cost. The constraints in the formulation assure that each trip is assigned to exactly one predecessor and one successor.

Freling et al. [18] compared the efficiency of several algorithms for the VSP, including the Hungarian algorithm [27], successive shortest path algorithm [14], and the minimum cost flow approach [6] and showed that auction based algorithms are the fastest and most stable on average. Since solving single-depot vehicle rescheduling problem is equivalent to solving $|G|$ vehicle scheduling problems, the auction algorithm was selected as our approach due to its excellent results for the VSP [18]. The auction method is also well suited for implementation on parallel machines [4], improving overall computational performance. This property is important for the vehicle rescheduling problem since it needs to be solved very quickly.

The next section discusses our sequential and parallel implementations of the auction-based algorithm for the SDVRSP.

5. AUCTION-BASED ALGORITHMS FOR SOLVING THE SDVRSP

Before describing the developed algorithms, we will introduce the basic concepts related to auction algorithms.

5.1. Auction Algorithms: An Introduction

The auction algorithm was originally proposed by Bertsekas [3] for the classical symmetric assignment problem. Given its computational performance, it was further developed for the shortest path, the asymmetric assignment problem, and the transportation problem [3]. In the classical symmetric assignment problem, we need to match n persons and n objects on an one-to-one basis. Let a_{ij} be the benefit of matching person i and object j. The objective function is to maximize the total benefit. In the auction algorithm, each object j has a price p_j, and this price is updated upwards as persons bid for their best object, that is, the object for which the corresponding benefit minus the price is maximal.

The auction algorithm is composed of two phases: bidding phase and assignment phase. In the bidding phase, every unassigned person looks for its "best" object; in the assignment phase, the object selects the person with the highest bid since it may receive more than one bid. Meanwhile, if some objects that have already been assigned to certain persons in a preceding iteration are now assigned to new persons, the persons who lost their objects are inserted into an unassigned set. After all the persons and objects are matched, the auction algorithm is terminated.

The combined forward and backward auction algorithm consists of forward and backward auction iterations. In the forward iteration, unassigned persons attempt to bid for objects, while, in the backward iteration, unassigned objects attempt to bid for persons. The combined auction algorithm has also been used for quasi-assignment problems [18]. The combined auction algorithm for these problems is similar to the combined algorithm for the classical assignment problem, except that the person and object which represent the depot do not participate in the bidding. In the combined auction algorithm for VSP, the person can be seen as the trip that is forward assigned, and the object can be seen as the trip that is backward assigned. The algorithms developed in the paper to solve SDVRSP are based on the combined auction algorithm developed by Freling et al. [18].

The performance of the auction algorithm is often improved by using the ϵ scaling of Bertsekas [3], where an ϵ is added to the prices, with ϵ gradually decreasing in subsequent iterations. When ϵ is less than $\frac{1}{n}$, where n is the number of elements to assign, an optimal solution is obtained. As suggested by Bertsekas and Castañon [4], a possible implementation of ϵ scaling is as follows: the integer benefits of a_{ij} are first multiplied by $n+1$ and the auction algorithm is applied with progressively lower values of ϵ, up to the point where ϵ becomes 1 or smaller. Using ϵ-scaling, the complexity of the algorithm is $O(nm, \log nC)$, where m is the number of possible assignments between pairs of elements and C is the maximum absolute benefit.

Freling et al. [18] describes the auction algorithm as follows: The value of a bid of trip i (or person i) for another trip j (or object j), which is a candidate for forward assignment, is denoted by $f_{ij} = a_{ij} - p_j$. The value of a bid of trip i for the depot is denoted by $f_{it} = a_{it}$. Let N be the set of all trips and M the arcs in the feasible network, respectively. Introduce π_j to denote the price of object j when the backward auction is conducted.

Algorithm A_QP: Auction Algorithm for the Quasi-Assignment Problem

STEP 1. Perform the forward auction algorithm for each trip $i \in N$ (or person i) which is currently not assigned to a trip j (or object j) or the depot.

STEP 2. Determine the trip or depot j_i with the maximum bid value $\beta_i = max\{f_{ij} | j : (i,j) \in M\}$. Determine also the second highest value $\gamma_i = max\{f_{ij} | j : (i,j) \in M, j \neq j_i\}$. If trip i (or person i) has only one arc $(i,j) \in M$, set $\gamma_i = -\infty$; If $j_i = t$ go to step 4.

STEP 3. Update the prices: $p_{j_i} = p_{j_i} + \beta_i - \gamma_i + \epsilon = a_{ij_i} - \gamma_i + \epsilon$, and $\pi_i = a_{ij_i} - p_{j_i}$. Update the assignments. If trip j_i was already backward assigned, then remove the previous assignment. Return to Step 1.

STEP 4. Update the price: $\pi_i = a_{it}$, update the assignment, and return to step 1.

The reverse auction procedure is similar, with bids for candidates for the forward assignments replaced by bids for candidates for the backward assignments [18].

5.2. Sequential Auction Algorithm for the VRSP

The sequential auction algorithm is based on the combined forward-backward auction algorithm developed by Freling et al. [18], considering the existence of several possible

feasible networks to be solved. The algorithm is described as follows:

Algorithm SA_VRSP: Sequential Auction Algorithm for VRSP

STEP 1. Based on the starting and ending times of trips and travel time between trips, apply the procedure described in Section 4.1 to build the set of all possible feasible networks. Calculate the costs for the compatible trip pairs and the total delay cost of each feasible network.

STEP 2. For each feasible network, apply the forward-backward combined auction algorithm [18] to find the minimum cost scheduling (i.e., assignment) for each feasible network as follows:

STEP 2.1. Set the initial prices to 0. Set the initial $\epsilon = (n+1) * C$, where C is the maximum absolute benefit.

STEP 2.2. Using current ϵ and prices from last iteration, conduct the bidding and assignment until all trips are both forward and backward assigned.

STEP 2.3. If $\epsilon \leq 1$, the auction algorithm for current feasible network terminates. Otherwise, set $\epsilon = 0.5 * \epsilon$ and clear the assignment, go to step 2.2.

STEP 3. Select the minimal operating and delay cost schedule as the solution.

5.3. Parallel Auction Algorithm

The system for the parallel synchronous algorithm is composed of an assignment processor and several bidding processors, where the assignment processor is in charge of determining the prices and making the assignment, and the bidding processor is in charge of conducting the bidding. We employ the Jacobi method to implement the parallel auction algorithm since this method needs less synchronization than the Gauss–Seidel method [4]. Suppose there are T bidding processors that conduct bidding. Then in the forward (backward) auction, the unassigned persons (objects) are partitioned into T subsets. Every bidding processor simultaneously conducts the bidding for a different subset. After bidding in each processor is completed, the results, including the partial assignment and prices of persons and objects for the specific subset, are sent to the assignment processor. When the assignment processor receives all results from the T bidding processors, it combines them to determine the new assignment and prices for all the unassigned persons and objects. If some objects (persons) that have already been assigned to some persons (objects) in a preceding iteration are now assigned to new persons (objects), the persons (objects) who lose the objects (persons) will be put into the set of unassigned persons (objects).

The new assignment information is sent back to the T bidding processors and the auction continues. After all the persons and objects are assigned, the auction algorithm is terminated. A method that partitions the unassigned trips will be presented later. Figure 3 illustrates the parallel synchronous implementation of the Jacobi method.

Since the forward–backward combined auction algorithm is used to solve SDVRSP, we have to determine if the auction should be forward or backward at each new iteration. The first iteration always uses a forward auction operation. We employ the method from [3] to refrain from switching between forward and backward auction until at least one more person-object pair has been added to the assignment.

To partition the unassigned trips and simultaneously conduct the bidding, a simple partitioning method is used to allocate each unassigned person (object) to the bidding

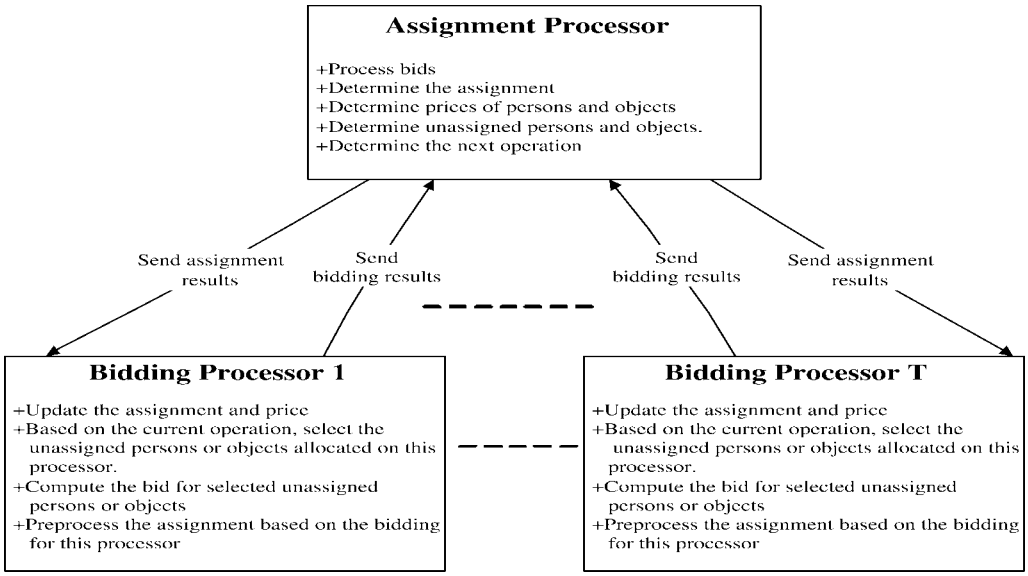

FIG. 3. Parallel synchronous auction algorithm.

processors. Every bidding processor is assigned an ID, in the range $0, 1, \ldots, T-1$. Considering that there are M unassigned persons (objects) stored in a list L, the unassigned persons (objects) for the bidding processor is defined by $Q[a_i] = i \bmod T, 0 \le i \le M-1$, where a_i is the i-th unassigned person (object) in list L, and $Q[a_i]$ is the designed bidding processor of person (object) a_i.

A preprocessing technique is also employed for accelerating the computation and reducing the data-handling traffic. Consider the following situation: If there is an excessive number of unassigned persons for each bidding processor (this typically happens in the early stages of auction algorithms), it is quite likely that several persons bid for the same object in the same bidding processor. It is possible to make partial assignments in each bidding processor rather than in the assignment processor, considering the most dominant person requesting an object in the bidding processor. After the partial assignment is carried out in each processor, only one person bids for the same object in this bidding processor. This partial assignment can reduce the amount of data sent to the assignment processor. Computational experiments show that this method significantly reduces the running time of the parallel implementation. The parallel algorithm is described as follows.

Algorithm PA_T_VRSP: Parallel Auction Algorithm with T Processors for the VRSP

STEP 1 and STEP 3 are the same as the corresponding steps in the sequential auction algorithm. Step 2 is as follows:

STEP 2. For each feasible network, apply the forward-backward combined parallel auction algorithm to find the minimum cost schedule for each feasible network as follows:

STEP 2.1. Set the initial prices to 0. Set the initial $\epsilon = (n+1) * C$. Send the information to bidding processors.

STEP 2.2a. Upon receiving current ϵ, assignment and prices from assignment processor, conduct the bidding for the persons or objects allocated on each processor. Then, make a partial assignment and send the results to the assignment processor.

STEP 2.2b. Based on the information received from the bidding processors, determine a minimum-cost assignment and the corresponding prices. If all persons and objects are assigned, go to step 2.3. Otherwise, send the assignment results to bidding processors and go to step 2.2a.

STEP 2.3. If $\epsilon \le 1$, the schedule for the current feasible network is optimal. Otherwise, set $\epsilon = 0.5 * \epsilon$, clear the assignment and send the information to bidding processor, and go to step 2.2.

6. COMMON FEASIBLE NETWORK APPROACH

As the computational time of the auction algorithm is highly dependent on initial prices, it is possible to improve the computational performance if a better set of initial prices is used. Bertsekas [3] states that "if the prices are near optimal, we expect that the number of iterations to solve the problem will be relatively small". In this section, a method to find near optimal prices, referred to as the *common feasible network*, is proposed.

Although the number of feasible networks can be large for a problem involving several trips, the total number of backup trip candidates is much smaller than the total trips. As a consequence, the large parts of the feasible networks are similar, except for the arcs related to the cut trip. In a feasible network, the incoming arc of the cut trip is one from a backup trip to the cut trip; this arc is different in different feasible networks since the backup trip is different. Meanwhile, the starting time of the cut trip is dependent on the backup trip. Accordingly, the outgoing arcs of the cut trip are also dependent on the backup trip, since arc cost and compatible trip pairs are based on the starting time of a trip. These arcs are also different in the different feasible networks. Therefore, if the cut trip and all the backup trips (they correspond to nodes here) and the arcs associated with them are removed from each feasible network, a sub-network that is common for all feasible networks is obtained. If the backup trip is from the depot, the node that denotes the depot is still kept in the common feasible network, since every node needs to be connected to the depot.

The common feasible network (CFN) can be obtained from the intersection of all possible feasible networks as follows: $CFN = \bigcap_{k \in K} G(k)$, where $|K| \ge 2$. Figure 4 illustrates the common feasible network obtained from the feasible networks presented in Figure 2.

The basic idea of the CFN approach is to solve the VRSP in two stages. First, the auction algorithm is applied to the CFN to obtain optimal prices of person and objects for this reduced network. Next, the auction algorithm is applied for all the feasible networks, using the final prices obtained in stage 1 as initial prices. The optimal prices obtained in stage 1 are likely to be closer to optimal and, therefore, these prices provide a good set of initial prices for the second stage. With better initial prices, the computation time for the second stage is significantly reduced.

To construct the feasible network corresponding to a backup trip k, $G(k)$ from CFN , the following nodes need

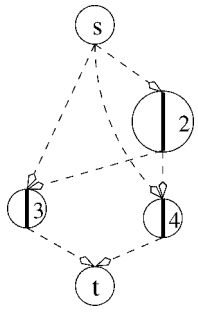

FIG. 4. The common feasible network for the Figure 2 example.

to be added to CFN: (i) the cut trip (b), and (ii) all backup trip candidates, defined by set K. Also, the following arcs are added to CFN: (i) arc (k,b); (ii) arcs (b,j) such that b and j are compatible trips, $j \in N$; (iii) arcs (c,j) such that c and j are compatible trips, $c \in K - \{k\}, j \in N$; and (iv) arcs $(s,c), c \in K$.

Appendix A gives an analysis why CFN improves computational performance towards solving VRSP, where we show that the computational time to solve VRSP using the CFN approach is expected to be less than without the use of the CFN. Results presented in Section 7 support this analysis. However, it cannot be claimed that the approach with the CFN is better on every VRSP instance due to the following reasons: (i) the upper bounds of the auction algorithms with and without the CFN approach are not tight; and (ii) in order to make this claim it is necessary to prove that the lower bound of the approach without the CFN is worse than the upper bound of the CFN approach. Both issues are presently being investigated by the authors.

The CFN based auction algorithm is described as follows.

Algorithm CFN_VRSP: Auction Algorithm for the VRSP Using CFN

STEP 1. Determine the possible backup trips.

STEP 2. Remove the cut trip and all backup trip candidates and the arcs associated with them. Build CFN.

STEP 3. Apply the sequential auction algorithm (or the parallel auction algorithm when $T \geq 2$ processors are used) described in Section 5.2 (or Section 5.3) for the CFN constructed in step 1.

STEP 4. For each possible backup trip, build the corresponding feasible network from CFN.

STEP 5. Using the optimal prices obtained in step 3 as the initial prices, apply the sequential auction (a parallel auction when T processors are used) algorithm for each feasible network.

STEP 6. Select the optimal solution with the minimal operating and delay cost.

We will denote *CFN_T_VRSP* as the parallel version of the above algorithm when $T \geq 2$ processors are used.

7. COMPUTATIONAL EXPERIMENTS

The main objective of computational experiments was to compare the performance of the developed algorithms, in terms of CPU time needed to obtain the optimum solution. Therefore, for convenience, we only included cost of reallocating vehicles (including the vehicle for each backup trip) and not the cost of delay to passengers on the disabled vehicle in the objective function for the SDVRSP.

Since the constraint matrix is totally unimodular, the solution of the linear relaxation for the VRSP provides an optimal solution. Nevertheless, solving linear relaxation may require longer times than the auction algorithm, because the latter was specially designed to solve the VSP. We used CPLEX 7.0 Network Optimizer to solve the linear relaxation of VRSP. CPU times of the linear relaxation and of the auction algorithms were compared for verification purposes.

The algorithms were implemented in C++ on Sun-Fire-880 Workstations, each of which with 2 UltraSPARCIII processors at 900 MHz, 4 GB of RAM and a Solaris 9 operating system. The communication protocol used for parallel implementation was developed based on the Socket/Stream protocol. The following notation is used in the tables to indicate the implemented algorithms:

(a) *CPLEX*: The use of CPLEX7.0 to solve the linear relaxation of VRSP;
(b) *SAV*: The sequential auction algorithm, *SA_VRSP*;
(c) *PA2*: The parallel auction algorithm using two processors, *PA_2_VRSP*;
(d) *PA4*: The parallel auction algorithm using four processors, *PA_4_VRSP*;
(e) *CFN*: The sequential auction algorithm with CFN, *CFN_VRSP*;
(f) *CFN2*: The parallel auction algorithm with CFN using two processors, *CFN_2_VRSP*;
(g) *CFN4*: The parallel auction algorithm with CFN using four processors, *CFN_4_VRSP*.

7.1. Experiments Configuration

The experiments were designed using Carpaneto et al.'s [9] random data generation method for VSP. Let $\rho_1, \rho_2, \ldots, \rho_v$ be *relief points* (i.e., points where trips can start or finish) of a transportation network. We generate them as uniformly random points on a (60×60) square and compute the corresponding travel times θ_{ρ_a,ρ_b} as Euclidean distances between relief points ρ_a and ρ_b. To simulate the trips, we generate for each trip T_j $(j = 1, \ldots, n)$ the starting and ending relief points, ρ'_j and ρ''_j, randomly selected from $\rho_1, \rho_2, \ldots, \rho_v$. The travel time between trips T_i and T_j is defined as $\theta_{\rho''_i\rho'_j}, \forall i,j$. The starting and ending times, s_j and e_j, of trip T_j are generated by considering two classes of trips: short trips and long trips. For short trips, s_j is a uniformly random integer in interval $(420, 480)$ with probability 0.15; in interval $(480, 1020)$ with probability 0.70; and in interval $(1020, 1080)$ with probability 0.15. Since ending time e_j for trip T_j must include a travel time between ρ'_j and ρ''_j, and dwell time at vehicle stops, we generate e_j as a uniformly random integer in $(s_j + \theta_{\rho'_j,\rho''_j} + 5, s_j + \theta_{\rho'_j,\rho''_j} + 40)$. For long trips, we assume they start and end at the same point, and the travel time depends on the length of resultant cycles and associated stops. Then we generate s_j as a uniformly random integer in $(300, 1200)$, and e_j as uniformly random integer in $(s_j + 180, s_j + 300)$. Costs c_{ij},

c_{si} and c_{jt} are defined to include travel time and waiting time; we used

- $c_{ij} = 10\theta_{i,j} + 2(s_j - e_i - \theta_{i,j})$, for all compatible pairs (T_i, T_j);
- $c_{si} = \lfloor 10$ (Euclidean distance between the depot and the starting point of trip $T_i)\rfloor + 2{,}000$, for trips from the depot to route T_i; and
- $c_{jt} = \lfloor 10$ (Euclidean distance between the depot and the ending point of trip $T_j)\rfloor + 2{,}000$, for trips from T_j to the depot.

In order to compare the computational efficiency of the different algorithms, we consider two situations: (i) the total number of trips is composed of a 40:60 combination of short and long trips (class L); (ii) the total number of trips is only composed of short trips (class S).

To evaluate the performance of the algorithms, we first generated a VSP problem and solved it. Then, a disruption was introduced so that an early short trip is chosen as the cut trip (trip T_b). Since in Carpaneto et al.'s [9] data generation method, long trips are cycles with same start and end locations, making it difficult to define a breakdown point with respect to start time, we only chose short trips as possible cut trips. We assumed that vehicles break down in the middle of the cut trip in time and distance. The arrival time of a backup vehicle candidate to the breakdown point is calculated as follows: (i) for a backup vehicle on a regular trip, the arrival time is the ending time of the current trip plus the travel time from the ending point to the breakdown point; and (ii) for a vehicle on a deadheading trip, the arrival time is the current time plus the travel time from its current location to the breakdown point. Euclidean distances are used as travel distances.

In real-life situations, the determination of backup trip candidates requires knowledge of vehicle capacity and common itineraries (see Section 4.1). However, in our experiments, the backup trip candidates are generated based on distances and travel times. In order to simplify the experiments and to better compare the algorithms, we assumed that K, the number of backup trips, is such that $K \in \{2, 3, 5, 10, 15, 20, 25, 30, 35, 40\}$ for test cases with more than 300 remaining trips. For test cases with 100 and 300 remaining trips, $K \in \{2, 3, 5, 10\}$ and $K \in \{2, 3, 5, 10, 15, 20\}$, respectively, since smaller-size problems have fewer backup vehicles. We sorted the existing vehicles using their arrival times to the breakdown point and selected the first K vehicles as the backup vehicle candidates. Note that these considerations are coherent with the main objective of the experiments, to compare the performance of the implemented algorithms. The procedure to find the backup vehicle candidates, described in Section 4.1, has already been tested in a real life application described in [35]. The new starting time, ST, of the cut trip is the arrival time of a particular backup vehicle at the breakdown point plus a 3− min start-up service time. The costs of the out-going arcs of the cut trip i are calculated using the expression defined for c_{ij}, considering the new ending time of trip i as $ST + \frac{e_i - s_i}{2}$.

7.2. Results

Tables 2 and 3 compare the performance of algorithms CPLEX, SAV, PA2, PA4, CFN, CFN2, and CFN4 for 10 instances of each problem in classes L and S, respectively. The first five columns display the number of remaining trips, the average original number of vehicles, the number of backup trips considered, the average number of new vehicles required to finish remaining trips, and the average optimal cost, respectively. Remaining columns show the average CPU times in seconds, excluding input and output time, for the seven algorithms. For example, if the remaining trips are 100 in class L, the number of vehicles needed for the original VSP is 28.6 on average over 10 instances (see first row in Table 2). If the number of remaining trips is 500 and there are two backup trip candidates in class L, the average number of vehicles needed for the VRSP is 122.1, and the optimal solution cost averaged 564,555 over 10 instances. Problems in class S are much more difficult to solve since this class contains only short trips, increasing the number of arcs in each possible feasible network.

The tables show that an increase in the number of available backup trips decreases optimal cost, characterizing a trade-off between CPU time and the optimal cost, defined by the number of possible backup trip alternatives being considered. Taking into consideration (i) the small differences in the average optimum cost between 2 and 40 backup trips alternatives for large problems, and (ii) the considerable increase in the average CPU time for these problems, it may be worthwhile to develop heuristics to prune the number of possible backup trip alternatives, specially for large trips. The idea is to select and solve the problem only for a representative subset, in a way that the feasible network that leads to the optimum solution is included, with a high probability, in the solution space.

An extra vehicle is needed to replace the disabled vehicle when the number of vehicles in the rescheduling problem turns out to be equal to the number of vehicles in the original scheduling. This is confirmed by the values presented in second and fourth column in Tables 2 and 3, in which the average number of vehicles to finish the trips is, in many cases, approximately equal to the average initial number of vehicles. The reason why an extra vehicle is sometimes needed in VRSP, as compared to VSP, is that in the vehicle rescheduling at the current system state, some deadheading vehicles are out of the depot at inconvenient positions in the network, while some trips need still to be finished.

The average CPU time for all algorithms is highly dependent on the problem size. The table shows that for small problems (100 remaining trips) all algorithms are extremely fast, solving problems, even with high values of $|K|$ in less than 1s CPU time. Comparing CPLEX and SAV, we can affirm that the auction-based algorithm is more efficient for all analyzed situations. To find the optimum solution, algorithm SAV reduces the required CPU time over the CPLEX by, on the average, 328.63%. The auction algorithm is more efficient for small and medium-sized problems (up to 1,000

TABLE 2. Computational results - class L.

Remaining trips	Initial # of vehicles	Backup trips	New # of vehicles	Objective value	Average CPU seconds						
					CPLEX	SAV	PA2	PA4	CFN	CFN2	CFN4
100	28.6	2	29.0	135,407	0.09	0.01	0.05	0.04	0.02	0.04	0.04
		3	28.8	134,516	0.13	0.02	0.08	0.05	0.03	0.06	0.07
		5	28.6	133,601	0.22	0.03	0.13	0.08	0.03	0.09	0.10
		10	28.6	133,511	0.45	0.07	0.25	0.18	0.07	0.19	0.18
300	76.0	2	76.6	356,042	0.88	0.21	0.18	0.17	0.23	0.22	0.17
		3	76.4	355,100	1.32	0.31	0.27	0.25	0.30	0.30	0.24
		5	76.2	354,189	2.20	0.51	0.48	0.41	0.45	0.45	0.36
		10	76.0	353,284	4.42	1.01	0.91	0.75	0.80	0.77	0.63
		15	76.0	353,256	6.65	1.49	1.32	1.13	1.19	1.01	0.91
		20	76.0	353,233	8.87	1.98	1.75	1.52	1.52	1.33	1.12
500	121.6	2	122.1	564,555	2.96	0.72	0.50	0.35	0.84	0.54	0.46
		3	122.0	564,071	4.41	1.07	0.75	0.53	1.12	0.66	0.56
		5	121.9	563,603	7.31	1.78	1.16	0.91	1.68	1.07	0.91
		10	121.6	562,277	14.55	3.50	2.29	1.82	3.05	1.88	1.64
		15	121.6	562,267	21.80	5.16	3.30	2.69	4.24	2.71	2.29
		20	121.6	562,232	29.05	6.78	4.32	3.56	5.34	3.28	2.99
		25	121.6	562,219	36.33	8.43	5.29	4.41	6.71	4.32	3.72
		30	121.6	562,214	43.58	10.08	6.34	5.24	8.36	5.27	4.53
		35	121.6	562,193	50.82	11.73	7.35	6.05	9.47	5.97	5.32
		40	121.6	562,179	58.09	13.38	8.35	6.92	10.78	6.80	5.95
700	165.2	2	165.8	767,241	6.71	1.73	0.82	0.70	2.06	1.10	0.93
		3	165.5	766,037	10.07	2.60	1.26	1.07	2.70	1.42	1.27
		5	165.3	765,181	16.82	4.30	2.07	1.78	4.07	2.09	1.96
		10	165.1	764,322	33.56	8.33	4.06	3.53	7.02	3.94	3.40
		15	165.1	764,213	50.41	12.47	6.10	5.22	10.06	5.71	4.75
		20	165.1	764,154	67.31	16.43	8.11	6.84	12.47	6.86	5.89
		25	165.1	764,151	84.16	20.33	10.03	8.39	15.49	8.49	7.49
		30	165.1	764,141	101.03	24.21	11.94	9.96	18.60	10.12	8.88
		35	165.1	764,135	117.96	28.05	13.97	11.51	20.90	11.55	9.94
		40	165.1	764,128	134.85	31.90	15.87	13.12	23.55	13.24	11.29
900	211.0	2	211.0	1,029,542	13.15	4.16	1.45	1.18	4.32	1.58	1.33
		3	210.9	1,028,997	19.81	6.23	2.17	1.74	5.49	2.22	1.75
		5	210.9	1,028,957	33.16	10.31	3.55	2.87	7.49	2.96	2.34
		10	210.7	1,028,106	66.46	20.62	7.10	5.73	13.70	5.79	4.59
		15	210.7	1,028,091	99.85	30.86	10.46	8.52	19.40	7.86	6.60
		20	210.7	1,028,066	133.31	41.07	13.84	11.35	25.33	10.03	8.53
		25	210.7	1,028,059	166.81	51.19	17.16	14.07	31.56	13.04	10.62
		30	210.7	921,768	200.22	61.16	20.45	16.79	40.07	16.48	13.79
		35	210.7	818,583	233.61	71.19	23.69	19.56	46.27	18.73	15.59
		40	210.7	818,576	267.11	81.17	26.99	22.32	49.70	19.84	16.56
1,100	253.6	2	253.5	1,218,215	21.54	10.99	2.84	2.03	10.54	2.96	2.32
		3	253.5	1,218,200	32.32	16.47	4.30	3.15	13.43	3.67	3.08
		5	253.5	1,218,162	54.12	27.23	7.02	5.35	19.36	6.43	4.51
		10	253.2	1,216,918	108.87	54.03	13.48	10.62	35.09	10.74	8.58
		15	253.2	1,216,906	163.53	80.40	19.82	15.73	49.48	15.64	12.01
		20	253.2	1,216,885	217.79	106.77	26.38	20.97	65.28	20.94	15.66
		25	253.1	1,216,458	271.89	133.17	32.80	26.11	81.47	25.77	19.69
		30	253.1	1,216,446	325.98	159.13	39.20	31.24	97.13	29.51	23.49
		35	253.1	1,216,446	380.22	185.34	45.63	36.43	113.97	34.85	27.73
		40	253.1	1,216,444	434.49	211.48	51.95	41.60	128.50	37.36	30.87
1,300	302.8	2	302.8	1,440,810	33.04	18.91	4.34	3.53	19.45	5.12	4.03
		3	302.6	1,439,972	49.38	28.31	6.51	5.31	25.22	7.11	5.39
		5	302.6	1,439,964	82.52	46.66	10.97	8.88	34.92	9.75	7.65
		10	302.5	1,439,139	165.47	91.86	21.64	17.50	63.65	18.79	15.00
		15	302.4	1,438,699	248.13	136.87	32.19	25.74	91.33	28.23	21.92
		20	302.4	1,438,686	331.27	182.19	42.68	34.25	128.65	39.78	31.82
		25	302.4	1,438,686	414.24	226.62	52.70	42.25	149.93	45.88	37.49
		30	302.4	1,438,674	496.81	271.51	62.88	50.52	181.64	58.80	43.91
		35	302.4	1,438,662	580.11	315.62	73.05	58.45	213.66	68.93	53.17
		40	302.4	1,438,662	662.99	359.45	82.80	66.21	232.24	72.95	56.47

TABLE 3. Computational results - class S.

Remaining trips	Initial # of vehicles	Backup trips	New # of vehicles	Objective value	Average CPU seconds						
					CPLEX	SAV	PA2	PA4	CFN	CFN2	CFN4
100	18.1	2	18.6	93,739	0.11	0.02	0.07	0.04	0.02	0.09	0.05
		3	18.3	92,280	0.17	0.03	0.10	0.06	0.04	0.10	0.08
		5	18.1	91,431	0.28	0.05	0.14	0.11	0.05	0.12	0.12
		10	18.1	91,358	0.56	0.11	0.32	0.24	0.10	0.31	0.18
300	48.9	2	49.7	249,782	1.24	0.37	0.32	0.20	0.46	0.41	0.31
		3	49.4	248,387	1.86	0.55	0.50	0.31	0.61	0.57	0.42
		5	49.3	247,896	3.09	0.90	0.82	0.53	0.86	0.70	0.60
		10	49.0	246,580	6.19	1.77	1.55	1.14	1.66	1.51	1.08
		15	48.9	246,133	9.28	2.61	2.19	1.72	2.36	2.12	1.59
		20	48.9	246,099	12.37	3.43	2.81	2.29	3.17	2.86	2.04
500	78.0	2	77.4	392,235	4.02	1.29	0.81	0.70	1.56	1.17	0.80
		3	77.8	391,764	6.03	1.96	1.21	1.03	2.23	1.47	1.26
		5	78.2	391,273	10.06	3.23	2.01	1.64	3.21	2.22	1.74
		10	78.5	389,579	20.15	6.25	3.84	3.17	5.58	4.07	3.15
		15	78.6	389,562	30.28	9.29	5.75	4.67	7.76	5.42	4.42
		20	78.3	389,512	40.41	12.30	7.59	6.15	9.88	6.93	5.58
		25	78.1	389,487	50.54	15.26	9.40	7.61	12.03	8.29	6.66
		30	78.0	389,478	60.66	18.17	11.27	9.05	13.55	9.96	7.80
		35	78.0	389,468	70.76	21.11	13.10	10.51	15.53	11.18	8.92
		40	78.0	389,462	80.89	23.98	14.98	11.98	17.33	12.90	10.28
700	106.7	2	106.5	551,275	9.70	3.43	1.82	1.43	4.03	2.31	1.74
		3	107.4	548,699	14.55	5.06	2.68	2.13	5.44	3.53	2.46
		5	107.3	547,782	24.12	8.28	4.49	3.54	8.19	4.96	3.87
		10	107.1	547,300	48.23	16.38	8.95	6.93	14.79	9.62	6.91
		15	107.1	547,294	72.35	24.70	13.45	10.52	21.79	13.34	10.40
		20	106.9	547,257	96.46	32.37	17.49	13.85	28.89	18.39	13.67
		25	106.8	547,241	120.64	39.99	21.69	17.15	33.97	23.25	16.21
		30	106.8	547,222	144.86	47.66	25.78	20.40	38.55	24.17	18.33
		35	106.8	547,220	169.05	55.59	29.99	23.78	45.92	29.41	22.13
		40	106.8	547,207	193.16	63.12	33.94	26.91	51.01	32.09	24.29
900	133.2	2	133.2	695,105	18.65	12.87	3.537	2.719	14.40	3.77	3.26
		3	133.4	695,058	28.02	19.11	5.273	4.094	18.66	5.83	4.14
		5	133.2	694,177	46.71	31.69	8.785	7.013	26.90	8.19	6.31
		10	133.1	693,700	93.55	63.83	17.505	13.541	48.51	16.92	11.47
		15	133.1	693,690	140.23	95.41	26.278	20.273	69.07	22.89	16.40
		20	133.0	693,241	186.72	126.29	34.673	27.147	90.84	29.26	21.67
		25	133.0	693,226	233.17	156.82	43.244	34.007	115.44	38.87	28.47
		30	133.0	693,223	279.89	187.21	51.384	40.215	139.14	47.29	36.09
		35	133.0	693,219	326.61	217.84	59.702	46.685	157.50	55.13	40.62
		40	133.0	693,219	373.07	248.46	67.652	52.978	181.35	62.11	47.72
1,100	163.5	2	164.6	885,735	31.07	27.26	6.02	5.02	28.44	7.08	5.29
		3	164.7	884,557	46.66	40.47	8.93	7.62	37.26	8.93	6.88
		5	164.6	884,499	77.59	66.55	14.74	12.23	54.10	12.81	10.62
		10	164.5	883,151	155.49	131.78	29.57	24.22	92.98	25.14	18.51
		15	164.5	883,141	233.24	196.42	44.03	36.10	138.58	34.53	27.66
		20	164.5	883,141	310.56	263.57	58.66	48.36	181.24	48.05	36.95
		25	164.5	883,084	387.71	328.93	73.51	60.74	228.69	63.46	47.76
		30	164.5	883,084	464.80	393.78	87.88	72.74	269.41	74.55	58.09
		35	164.4	883,081	541.97	458.62	102.21	84.39	327.80	92.55	70.82
		40	164.4	883,079	619.06	523.49	116.60	96.13	370.36	108.62	83.69
1,300	190.4	2	190.4	999,106	46.03	43.52	9.38	7.65	48.00	10.76	9.04
		3	190.4	998,612	69.15	65.45	13.93	11.62	63.03	13.68	11.33
		5	190.4	998,565	115.23	108.57	22.98	19.25	90.96	22.50	16.67
		10	190.1	996,805	229.85	215.81	45.70	37.87	158.85	40.24	31.54
		15	190.1	996,758	345.76	321.79	68.32	56.59	227.91	56.85	44.84
		20	190.0	996,328	461.09	425.71	91.21	74.79	292.17	75.83	60.51
		25	189.9	995,883	576.78	531.14	113.51	93.21	353.78	94.55	74.99
		30	190.0	995,876	693.54	639.36	136.19	112.51	433.50	115.12	93.03
		35	190.0	995,875	810.69	747.78	159.09	131.04	502.44	132.42	106.24
		40	190.0	995,875	927.38	854.38	181.83	150.13	569.10	145.46	119.42

remaining trips). As the problem size increases, the auction algorithm slightly reduces its efficiency. For large-sized problems, the CPU time differences between CPLEX and SAV are much smaller for class S than for class L. There are much more arcs in class S than in class L. It appears that the auction algorithm tends to be more influenced by the underlying network density than the network simplex used in the CPLEX method. In summary, we can conclude that the auction based algorithms are computationally efficient in solving the SDVRSP.

Computational results show that parallel processing does not improve CPU times for small problems. The reason for this due to the required communication cost among the multiple processors. However, for large problems, the communication time decreases its relevance and the slightly increased time is compensated by the fast processing time of the auction algorithm. The parallel auction with common feasible network using 4 processors is, on average, the most efficient for large problem sizes, in terms of the number of remaining trips and possible backup trips. Even for very large problems, such as 1300 remaining trips and 40 backup trips, this algorithm provides optimal solution in an acceptable amount of time (around 119s, considering only short trips). The parallel algorithms with the common feasible network (CFN2 and CFN4) become more efficient for both classes as problem size increases (more remaining trips and more backup trips).

Figures 5 and 6 present a pairwise comparison on the average relative CPU time percentage, referred to as CPU_{per}, considering SAV as the comparison basis. Figure 5 compares the algorithms in terms of the number of remaining trips, while Figure 6 compares them in terms of the number of backup trips. CPU_{per} is computed as follows:

$$CPU_{per} = 100 \times \frac{CPU_A}{CPU_{SAV}}$$

where CPU_{SAV} is the CPU time required by the algorithm SAV to find the optimum solution of SDVRSP, and CPU_A is the CPU time required by the algorithm being compared, say algorithm A. For problems with more than 300 remaining trips, the use of parallelism results in significant reductions on the average CPU time required to solve the problems. Reductions in the CPU times are more significant for larger problems and more backup trips.

Table 4 indicates the main explanation for the good performance of the CFN-based algorithms. This table gives the average number of iterations required by the sequential auction algorithm with and without using CFN to solve 10 instances of each problem for both the classes. In this table, we report the relative percentage of the number of iterations for CFN-based algorithms, defined by

$$I_{per} = 100 \times \frac{I_{CFN}}{I}$$

where I is the number of iterations without using CFN and I_{CFN} is the number of iterations using the CFN approach.

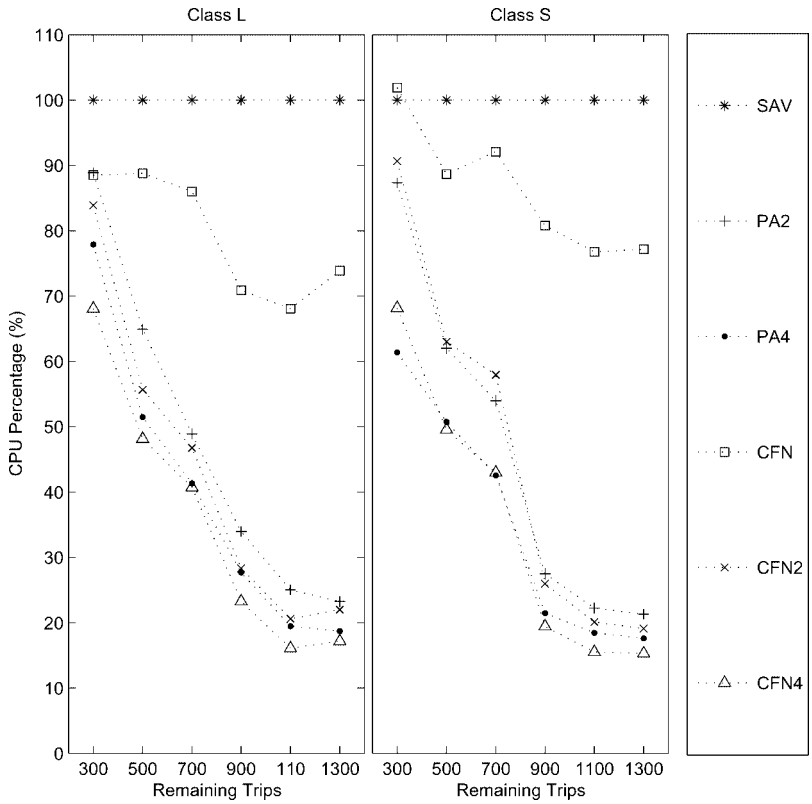

FIG. 5. Average CPU time percentage vs. the number of remaining trips.

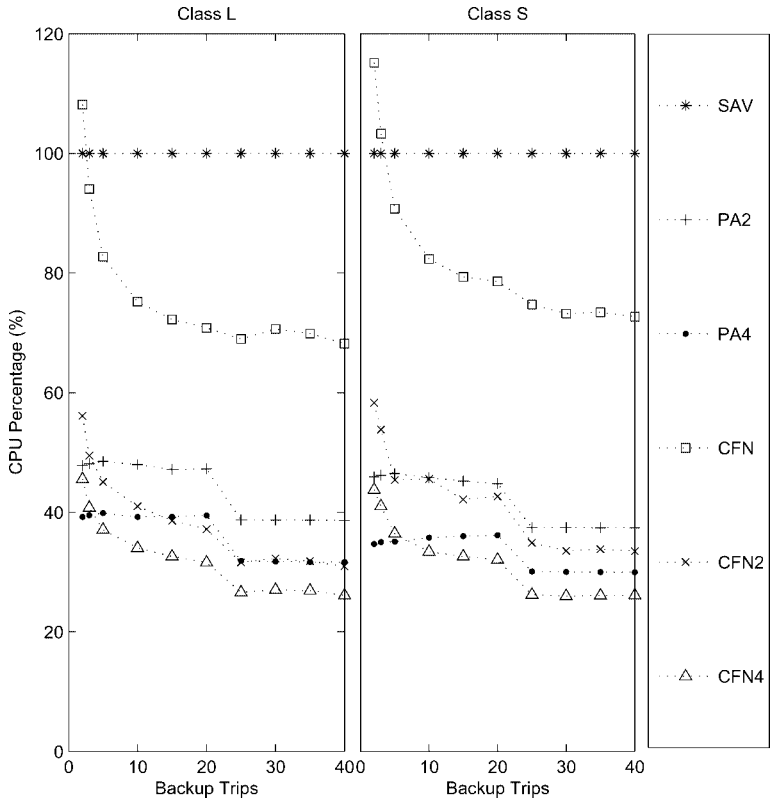

FIG. 6. Average CPU time percentage vs. the number of backup trips.

For class L, CFN algorithms requires, on average, 74.91% the number of iterations in comparison with non-CFN algorithms. For class S, the percentage was, on average, 84.78%. However, note that for very small number of backup trips (especially 2 and 3), the CFN algorithm does not work better, on average, than algorithms that do not use CFN (see plots in Fig. 6). For example, for problems with 100 remaining trips and 2 backup trips, the CFN algorithm requires, on average, an 11.30% increase in the number of iterations. This is because of the ϵ-scaling method used in the second stage of CFN approach to obtain convergent stability. As a consequence, the optimal prices from stage 1 are near optimal prices for the first subproblem in stage 2. In our implementations, the ratio, say R, between the average number of iterations in stage 2 and the total number of iterations in stage 1 is in the range [0.55, 0.82]. If we consider that the average number of iterations without using CFN is given by $|K|L$, where L is the average number of iterations for each $k \in K$, then the number of iterations for CFN will be approximately given by $L + |K|RL$. For $|K| = 2$, the average number of iterations will be $2L$ without using CFN, and about $L + 2 \times 0.82 \times L = 2.64L$ when using CFN, resulting in an increase in the number of iterations. However, for $|K| = 40$, the corresponding values are $40L$ and $L + 40 \times 0.83 \times L = 34.2L$, respectively, generating a smaller I_{per}.

Table 5 presents the speedup factors for algorithms with and without using CFN. Since parallel implementations are not faster for small-size problem, we only present speedup factors for problems with more than 300 remaining trips. We can see that when the problem becomes larger, parallel implementations provide significant CPU time reduction. For example, when there are 1,100 remaining trips in class L, a parallel auction algorithm with two processors is, on average, four times faster than its sequential counterpart. This is possibly explained as follows. For a large problem, the CPU cache is not sufficient for the sequential algorithm, resulting in swapping between the CPU cache and slower RAM, increasing significantly the CPU time of sequential algorithms to find the optimum solution. Since in parallel implementations, every processor conducts computations on a subset of the whole problem, the use of CPU cache is smaller than for its sequential counterpart, mitigating swapping between CPU cache and RAM.

8. CONCLUSIONS

This paper introduces and models the single depot vehicle rescheduling problem and presents several algorithms to solve this problem. The solution approach is based on (i) the generation of all possible feasible networks obtained when a trip is disrupted, and (ii) the application of auction algorithms for solving the resultant vehicle scheduling problem. In addition, the use of parallel processing and the common feasible network (CFN) approach improves the speed of the auction algorithms.

The CFN method performs the auction algorithm in two stages. In the first stage, the algorithm is carried out for a

TABLE 4. Comparison of the number of iterations for the sequential auction algorithm with and without using the CFN.

Remaining trips	Backup trips	Class L			Class S		
		Without CFN	With CFN	I_per (%)	Without CFN	With CFN	I_per (%)
100	2	4,370	4,864	111.30	5,543	6,220	112.21
	3	6,562	6,390	97.39	8,314	8,311	99.97
	5	11,015	9,800	88.97	13,909	12,185	87.61
	10	22,092	18,846	85.31	28,249	24,862	88.01
300	2	20,622	21,993	106.65	28,079	35,273	125.62
	3	30,871	28,488	92.28	42,162	46,617	110.57
	5	51,082	41,530	81.30	69,619	65,664	94.32
	10	101,788	72,618	71.34	137,310	125,162	91.15
	15	151,998	108,810	71.59	202,778	176,913	87.24
	20	202,247	140,121	69.28	266,421	238,173	89.40
500	2	41,682	46,756	112.17	59,180	70,764	119.58
	3	62,177	61,023	98.14	89,561	99,875	111.52
	5	103,954	89,950	86.53	148,339	142,688	96.19
	10	206,484	160,767	77.86	288,727	243,909	84.48
	15	306,520	225,004	73.41	429,713	339,443	78.99
	20	405,176	287,815	71.03	569,305	430,656	75.65
	25	504,779	358,126	70.95	706,917	523,377	74.04
	30	605,166	444,217	73.40	843,121	591,957	70.21
	35	704,922	508,677	72.16	979,928	677,673	69.16
	40	805,760	583,449	72.41	1,113,859	759,086	68.15
700	2	69,545	78,666	113.1	108,024	125,506	116.18
	3	104,537	101,265	96.87	159,770	167,659	104.94
	5	173,789	149,187	85.84	262,051	248,179	94.71
	10	339,789	256,528	75.50	519,543	444,486	85.55
	15	509,738	364,453	71.50	782,475	648,405	82.87
	20	674,101	456,892	67.78	1,027,346	859,393	83.65
	25	837,457	569,597	68.02	1,27,0495	1,012,623	79.70
	30	1,000,397	680,809	68.05	1,514,442	1,147,414	75.76
	35	1,162,426	772,101	66.42	1,766,807	1,363,320	77.16
	40	1,324,799	873,473	65.93	2,008,187	1,515,446	75.46
900	2	100,059	98,261	98.20	158,272	179,998	113.73
	3	149,861	124,123	82.83	235,163	234,565	99.75
	5	248,334	168,303	67.77	391,965	341,189	87.05
	10	495,989	303,054	61.10	789,359	617,562	78.24
	15	742,760	424,118	57.10	1,182,914	869,626	73.52
	20	988,506	553,871	56.03	1,568,450	1,15,3300	73.53
	25	1,232,593	688,691	55.87	1,949,303	1,491,448	76.51
	30	1,474,668	868,447	58.89	2,329,861	1,832,350	78.65
	35	1,717,956	1,010,284	58.81	2,711,335	2,091,371	77.13
	40	1,961,265	1,086,438	55.39	3,090,942	2,435,687	78.80
1,100	2	142,379	133,731	93.93	231,976	241,933	104.29
	3	213,160	168,960	79.26	344,989	318,412	92.30
	5	351,301	241,161	68.65	570,423	462,232	81.03
	10	697,461	436,318	62.56	1,13,0573	787,783	69.68
	15	1,040,391	612,454	58.87	1,683,398	1,177,799	69.97
	20	1,381,852	806,060	58.33	2,254,404	1,556,979	69.06
	25	1,724,343	1,002,762	58.15	2,812,030	1,986,139	70.63
	30	2,063,246	1,198,551	58.09	3,370,602	2,367,075	70.23
	35	2,405,397	1,400,586	58.23	3,927,923	2,871,917	73.12
	40	2,745,713	1,583,387	57.67	4,485,906	3,359,065	74.88
1,300	2	180,424	182,643	101.23	293,920	318,473	108.35
	3	271,627	237,701	87.51	439,263	417,065	94.95
	5	449,410	327,699	72.92	731,269	605,347	82.78
	10	886,438	596,912	67.34	1,458,431	1,064,770	73.01
	15	1,318,516	862,114	65.39	2,179,992	1,521,593	69.80
	20	1,753,211	1,220,468	69.61	2,888,851	1,973,755	68.32
	25	2,178,847	1,427,534	65.52	3,601,806	2,399,003	66.61
	30	2,610,488	1,726,665	66.14	4,322,864	2,912,496	67.37
	35	3,035,719	2,033,132	66.97	5,042,982	3,367,081	66.77
	40	3,457,568	2,203,216	63.72	5,753,194	3,842,732	66.79

TABLE 5. Speedup factors.

	Class L						Class S					
Remaining trips	300	500	700	900	1100	1300	300	500	700	900	1100	1300
SAV/PA2	1.13	1.54	2.05	2.94	4.00	4.30	1.15	1.61	1.85	3.64	4.49	4.69
SAV/PA4	1.29	1.94	2.42	3.61	5.14	5.34	1.64	1.97	2.35	4.66	5.42	5.68
CFN/CFN2	1.06	1.59	1.83	2.49	3.29	3.34	1.13	1.41	1.59	3.10	3.81	4.01
CFN/CFN4	1.30	1.84	2.11	3.03	4.21	4.28	1.50	1.78	2.14	4.14	4.93	5.01

reduced network that excludes the cut trip and the backup trip candidates (served by possible backup vehicles). Then an assignment close to the original one is obtained since this reduced network is very similar to the original feasible networks. In the second stage, the cut trip and backup trips are introduced to reconstruct feasible networks. The auction algorithms are applied again, using the initial prices obtained in the first stage. The second stage is performed for all candidate backup vehicles.

From extensive computational experiments performed on randomly generated data, the following observations were made:

1. For small problems (less than 300 remaining trips), both sequential and parallel implementations are fast. The sequential algorithm, without using parallel processing, provided the solution with the smallest CPU time on average, because the added communication time between processors was not offset by the computational efficiency gained with parallel algorithms.
2. For large problems (more than 300 remaining trips), the parallel algorithms using CFN very often provided the best performance on average. On average, the fastest algorithm was the parallel auction using CFN and four processors.
3. While CFN-based algorithms do not reduce computation time when candidate backup trips are two or three, they considerably mitigate computational efforts when backup trips are more than five. The good performance of CFN-based algorithms is explained by the smaller number of iterations required to find the optimal solution. On average, the use of the CFN approach requires around 79.85% of the iterations required by the auction algorithm that does not use CFN.

In summary, it can be concluded that the solution approaches developed in this paper are computationally efficient for their application to real-time schedule recovery problems.

We included two major assumptions in this study: (i) only the cut trip can suffer delays; and (ii) there is no restriction on the number of rescheduled trips. These assumptions may not be true for some applications. In some cases, a vehicle breakdown may also delay other trips (e.g., when the starting point of the next trip that the breakdown vehicle is scheduled to cover is too far from the depot and other vehicles). As a next step, a trip cancellation strategy is being introduced to handle such cases. The research team is also planning to include a strategy to limit the number of trips that can be rescheduled. Moreover, to reduce the travel delay for some major transit routes, a transit signal priority technique is being combined with this rescheduling algorithm.

Acknowledgments

This paper was written while the third author visited the ATLAS Center at the University of Arizona. Also, the authors like to express their sincere thanks to the anonymous referees for their valuable suggestions.

APPENDIX: ANALYSIS OF THE CFN-BASED ALGORITHMS

The main objective of this analysis is to determine an upper bound for the total number of operations required by the auction algorithms with and without using the CFN approach. It is expected that the upper bound using CFN is no greater than the upper bound without using CFN to solve the SDVRSP.

Let the maximal arc benefit in the network be C. Let $N = \{1, 2, \ldots n\}$ be the set of persons, and let $O = \{1, 2, \ldots n\}$ be the set of objects ($|N| = |O|$). Two different cases, with and without CFN, will be considered.

In this analysis, we make the following assumptions: (i) All arc costs, a_{ij}, are integer and nonnegative; (ii) The initial prices of all objects are set to 0; and (iii) For each node in the network, there is a list that stores the information of its adjacent arcs.

The major operation of the auction algorithm is to scan the adjacency list to determine the best and the second best object prices. In the sequel, the term "operation" means scanning an arc in the list and updating an object price if necessary [2].

Case 1: Without Common Feasible Network

Let the final optimal price of object j be p_j^*. Since finding the optimal price is the objective of the dual assignment problem, the auction algorithm terminates when the optimal prices are obtained [3]. In the auction algorithm, the price increment is defined as the profit of the best object minus the profit of the second best object plus an increment ϵ. Bertsekas [3] showed that if $\epsilon < 1/n$, the final assignment is optimal.

Therefore, if object j receives a bid, the minimal increment is ϵ. Hence, the total number of iterations in which the object j receives a bid is no more than $(p_j^* - 0)/\epsilon = p_j^*/\epsilon$. Consider an iteration in which object j increases its price. The worst

case occurs when all persons bid object j. In this situation, the complexity for person i's bid is $d(i)$, where $d(i)$ is the out-degree of person i, since the major operation involves scanning the whole adjacency list of node i. Thus, the total number of operations for object j at any iteration is, at most, $\sum_{i\in N} d(i) = Y$.

Since there are at most p_j^*/ϵ iterations conducted for the object j, the number of operations for object j are at most Yp_j^*/ϵ. Since there are n objects, the total number of operations, in the worst-case, is $Y(p_1^* + p_2^* + \cdots + p_n^*)/\epsilon$.

Case 2: With Common Feasible Network

Here, a two-stage mechanism is employed. In stage 1, let the initial prices of all objects be 0. Let N_1 be the set of persons and O_1 be the set of objects that are removed in stage 1 ($|N_1| = |O_1| = n_1$ and $n_1 \ll n$) to define the common feasible network. In VRSP, N_1 and O_1 include the persons and objects that represent the cut trip and all backup trip candidates. The network composed by $N - N_1$ persons and objects is defined as the CFN. Some associated objects of person k may belong to O_1. Let the number of associated objects of person k belonging to O_1 be $d_1(k)$, and the number of associated objects of person k belonging to $O - O_1$ be $d_2(k)$. Obviously, $d(k) = d_1(k) + d_2(k)$. Figure 7 illustrates these definitions.

The auction algorithm is first applied for the CFN, and optimal prices are obtained for this reduced network. In stage 2, we set the initial prices of objects in $O - O_1$ as the corresponding final prices from stage 1, and set prices of objects in O_1 as 0. The auction algorithm is then applied for all persons and objects.

STAGE 1. Auction algorithm for common feasible network

Analysis here is similar to case 1. However, there are $n-n_1$ persons and objects respectively. The set of objects is $\{n_1 + 1, n_1 + 2, \ldots, n\}$.

Let the final optimal price of object j be $\bar{p}_j^*$. There are $n-n_1$ objects, so the total number of operations, in the worst-case, is $Y_1(\bar{p}_{n_1+1}^* + \bar{p}_{n_1+2}^* + \cdots + \bar{p}_n^*)/\epsilon$, where $Y_1 = \sum_{i\in(N-N_1)} d_2(i)$ is the total number of arcs connecting the $n - n_1$ persons in $N - N_1$ to the $n - n_1$ objects in $O - O_1$.

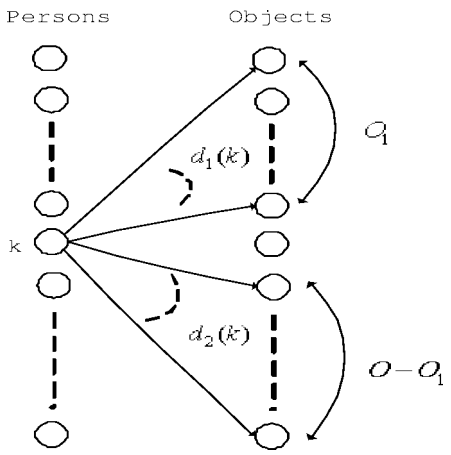

FIG. 7. Example of different objects in the CFN.

STAGE 2. Including the persons in N_1 and objects in O_1

The final optimal price for object j is p_j^*. The auction algorithm terminates when this optimal price is obtained. Two kinds of objects need to be considered. One kind belongs to O_1, whose initial prices are 0. The other belongs to $O - O_1$, whose initial prices are $\bar{p}_j^*$, obtained in stage 1.

For objects in O_1, the analysis is similar to case 1, therefore the total number of operation in the worst-case can be expressed as $Y(p_1^* + p_2^* + \cdots + p_{n_1}^*)/\epsilon$.

For objects j in $O - O_1$, the price needs to be increased by $p_j^* - \bar{p}_j^*$. It should be noted that we assume $p_j^* \geq \bar{p}_j^*$ here. If object j receives a bid, the minimal increment is ϵ. Total number of iterations in which the object j receives a bid is no more than $(p_j^* - \bar{p}_j^*)/\epsilon$. Therefore, the number of operations for object j is at most $Y(p_j^* - \bar{p}_j^*)/\epsilon$. Since there are $n - n_1$ objects being considered, the total number of operations is $Y(p_{n_1+1}^* - \bar{p}_{n_1+1}^* + p_{n_1+2}^* - \bar{p}_{n_1+2}^* + \cdots + p_n^* - \bar{p}_n^*)/\epsilon$.

Therefore, the total number of operations for all objects is at most

$$Y\big(p_1^* + p_2^* + \cdots + p_{n_1}^*\big)/\epsilon + Y\big(p_{n_1+1}^* - \bar{p}_{n_1+1}^* + p_{n_1+2}^* - \bar{p}_{n_1+2}^* + \cdots + p_n^* - \bar{p}_n^*\big)/\epsilon.$$

This expression can be reduced to

$$Y\big(p_1^* + p_2^* + \cdots + p_n^* - \big(\bar{p}_{n_1+1}^* + \bar{p}_{n_1+2}^* + \cdots + \bar{p}_n^*\big)\big)/\epsilon.$$

Complexity Analysis

For case 1, let the total number of operations be $F = Y(p_1^* + p_2^* + \cdots + p_n^*)/\epsilon$. For case 2, let the total number of operations in stage 1 be $F_1 = Y_1(\bar{p}_{n_1+1}^* + \bar{p}_{n_1+2}^* + \cdots + \bar{p}_n^*)/\epsilon$. Let the total number of operations in stage 2 be $F_2 = Y(p_1^* + p_2^* + \cdots + p_n^* - (\bar{p}_{n_1+1}^* + \bar{p}_{n_1+2}^* + \cdots + \bar{p}_n^*))/\epsilon$.

Let K be the set of backup trip candidates. There are $|K|$ backup trips; therefore for case 1, the regular auction needs to be applied $|K|$ times. The total number of operations is approximately $|K|F$. Using the two-stage method, stage 1 needs to be conducted only once, and stage 2 needs to be performed $|K|$ times. Thus, the total operation is approximately $F_1 + |K|F_2$. Our objective is to show that $|K|F > F_1 + |K|F_2$. This means that

$$K > \frac{F_1}{F - F_2}.$$

Observe that

$$\frac{F_1}{F - F_2} = \frac{Y_1\big(\bar{p}_{n_1+1}^* + \bar{p}_{n_1+2}^* + \cdots + \bar{p}_n^*\big)}{Y\big(\bar{p}_{n_1+1}^* + \bar{p}_{n_1+2}^* + \cdots + \bar{p}_n^*\big)} = \frac{Y_1}{Y}.$$

Since $\frac{Y_1}{Y}$ is less than 1, and $|K| \geq 2$, $|K|F > F_1 + |K|F_2$ always holds. This means that we can expect the CFN

approach to reduce the number of operations required by the auction algorithm to find the optimum solution.

REFERENCES

[1] F. Baita, R. Pesenti, W. Ukovich, and D. Favaretto, A comparison of different solution approaches to the vehicle scheduling problem in a practical case, Comput Oper Res 27 (2000), 1249–1269.

[2] D. Bertsekas, Linear network optimization: Algorithms and codes, The MIT Press, Massachusetts, 1991.

[3] D. Bertsekas, Auction algorithms for network flow problems: a tutorial introduction, Comput Optim Appl 1 (1992), 7–66.

[4] D. Bertsekas and D. Castañon, Parallel synchronous and asynchronous implementations of the auction algorithm, Parallel Comput 17 (1991), 707–732.

[5] L. Bodin and B. Golden, Classification in vehicle routing and scheduling, Networks 11 (1981), 97–108.

[6] U. Bokinge and D. Hasselstrom, Improved vehicle scheduling in public transport through systematic changes in the time-table, Eur J Oper Res 5 (1980), 388–395.

[7] R. Borndoerfer, A. Loebel, and S. Weider, A bundle method for integrated multi-depot vehicle and duty scheduling in public transit, Proceedings of the 9th international conference on computer-aided scheduling of public transport, In CDROM, San Diego, California, USA, 2004.

[8] P.M. Carlson, Exploiting the opportunities of collaborative decision making: A model and efficient solution algorithm for airline use, Transport Sci 34 (2000), 381–393.

[9] G. Carpaneto, M. Dell'Amico, M. Fischetti, and P. Toth, A branch and bound algorithm for the multiple depot vehicle scheduling problem, Networks 19 (1989), 531–548.

[10] A. Ceder, Urban transit scheduling: framework, review and examples, J Urban Plann Develop 128 (2002), 225–244.

[11] C.K. Chiu, J.H.M. Lee, H.F. Leung, and Y.W. Leung, A constrained-based interactive train rescheduling tool, Constraints 7 (2002), 167–198.

[12] J.R. Daduna and J.M. Paixão, Vehicle scheduling for public mass transit—an overview, Proceedings of the 6th International conference on computer-aided scheduling of public transport, Boston, MA, 1995, pp. 76–90.

[13] S. de Groot and D. Huisman, Vehicle and crew scheduling: Solving large real-world instances with an integrated approach, Proceedings of the 9th international conference on computer-aided scheduling of public transport, In CDROM, San Diego, California, USA, 2004.

[14] M. Dell'Amico, Una nuova procedura di assegnamento per il vehicle scheduling problem, Ricerca Operativa 5 (1989), 13–21.

[15] M. Dell'Amico, M. Fischetti, and P. Toth, Heuristic algorithms for the multiple depot vehicle scheduling problem, Management Science 39 (1993), 115–125.

[16] T.C. Du, E.Y. Li, and D. Chou, Dynamic vehicle routing for online b2c delivery, Omega-Int J Manage Sci 33 (2005), 33–45.

[17] B. Fleischmann, S. Gnutzmann, and E. Sandvoss, Dynamic vehicle routing based on online traffic information, Transport Sci 38 (2004), 420–433.

[18] R. Freling, A.P.M. Wagelmans, and J.M. Paixão, Models and algorithms for single-depot vehicle scheduling, Transport Sci 35 (2001), 165–180.

[19] M. Gendreau and J.-Y. Potvin, Dynamic vehicle routing and dispatching, Fleet management and logistics, T. Crainic and G. Laporte (Editors), Kluwer, New York, 1998, pp. 115–126.

[20] G. Ghiani, F. Guerriero, G. Laporte, and R. Musmanno, Real-time vehicle routing: solution concepts, algorithms and parallel computing strategies, Eur J Oper Res 151 (2003), 1–11.

[21] A. Haghani and S. Jung, A dynamic vehicle routing problem with time-dependent travel time, Comput Oper Res 32 (2005), 2959–2986.

[22] D. Huisman, R. Freling, and A.P.M. Wagelmans, A robust solution approach to the dynamic vehicle scheduling problem, Transport Sci 38 (2004), 447–458.

[23] S. Ichoua, M. Gendreau, and J.-Y. Potvin, Diversion issues in real-rime vehicle dispatching, Transport Sci 34 (2000), 426–438.

[24] R. Jonker and A. Volgenant, Improving the Hungarian assignment problem, Oper Res Lett 5 (1986), 171–176.

[25] L. Lettovský, Airline operations recovery: An optimization approach, Ph.D. thesis, Georgia Institute of Technology, USA, 1997.

[26] J. Medanic and M.J. Dorfman, Efficient scheduling of traffic on a railway line, J Optim Theory Appl 115 (2002), 587–602.

[27] J.M. Paixão and I. Branco, A quasi-assignment algorithm for bus scheduling, Networks 17 (1987), 249–269.

[28] H.N. Psaraftis, Dynamic vehicle routing: Status and prospects, Annals Oper Res 61 (1995), 143–164.

[29] J.M. Rosenberger, E.L. Johnson, and G.L. Nemhauser, Rerouting aircraft for airline recovery, Transport Sci 37 (2003), 408–421.

[30] T. Song and L. Zhou, A new algorithm for the quasi-assignment problem, Ann Oper Res 37 (1990), 205–223.

[31] M.Z. Spivey and W.B. Powell, The dynamic assignment problem, Transport Sci 38 (2004), 399–419.

[32] D. Teodorović and G. Stojković, Model to reduce airline schedule disturbances, J Transport Eng ASCE 121 (1995), 324–331.

[33] C.G. Walker, J.N. Snowdon, and D.M. Ryan, Simultaneous disruption recovery of a train timetable and crew roster in real time, Comput Oper Res 32 (2005), 2077–2094.

[34] J. Yang, P. Jaillet, and H.S. Mahmassani, Real-time multivehicle truckload pickup and delivery problems, Transport Sci 38 (2004), 135–148.

[35] J.-Q. Li, D. Borenstein, and P.B. Mirchandani, A decision support system for the single-depot vehicle rescheduling problem, Comput Oper Res 34 (2007), 1008–1032.