Models and Algorithms for Single-Depot Vehicle Scheduling

3 authors, including:

Albert P. M. Wagelmans
Erasmus University Rotterdam
116 PUBLICATIONS **3,794** CITATIONS

José Paixão
University of Lisbon
46 PUBLICATIONS **1,374** CITATIONS

Models and Algorithms for Single-Depot Vehicle Scheduling

Richard Freling • Albert P. M. Wagelmans • José M. Pinto Paixão

Econometric Institute, Erasmus University, Rotterdam, The Netherlands
DEIO, Universidade de Lisboa, Lisbon, Portugal

Vehicle scheduling is the process of assigning vehicles to a set of predetermined trips with fixed starting and ending times, while minimizing capital and operating costs. This paper considers modeling, algorithmic, and computational aspects of the polynomially solvable case in which there is a single depot and vehicles are identical. A quasiassignment formulation is reviewed and an alternative asymmetric assignment formulation is given. The main contributions of the paper are a new two-phase approach which is valid in the case of a special cost structure, an auction algorithm for the quasiassignment problem, a core-oriented approach, and an extensive computational study. New algorithms are compared with the most successful algorithms for the vehicle-scheduling problem, using both randomly generated and real-life data. The new algorithms show a significant performance improvement with respect to computation time. Such improvement can, for example, be very important when this particular vehicle-scheduling problem appears as a subproblem in more complex vehicle- and crew-scheduling problems.

The single-depot vehicle-scheduling problem (SDVSP) is defined as follows: Given a depot and a set of trips with fixed starting and ending times, and given the travelling times between all pairs of locations, find a feasible minimum-cost schedule such that (1) each trip is assigned to a vehicle, and (2) each vehicle performs a feasible sequence of trips. All the vehicles are supposed to be identical. A schedule for a vehicle is composed of *vehicle blocks,* where each block is a departure from the depot, the service of a feasible sequence of trips, and the return to the depot. The cost function is usually a combination of vehicle capital (fixed) and operational (variable) cost. The SDVSP is well known to be solvable in polynomial time.

Overviews of algorithms and applications for the SDVSP and some of its extensions can be found in Daduna and Paixão (1995) and in Desrosiers et al. (1995). Several network-flow-type algorithms have been proposed for the SDVSP in the literature. In particular, the SDVSP has been formulated as a linear assignment problem, a transportation problem, a minimum-cost flow problem, a quasiassignment problem, and a matching problem. We briefly discuss the most relevant algorithms. Let n be the number of trips to be covered by vehicles. Paixão and Branco (1987) propose an $O(n^3)$ quasiassignment algorithm which is specially designed for the SDVSP. Computational results indicate that this approach significantly outperforms approaches based on transportation and linear-assignment models. The quasiassignment algorithm is a modified version of the Hungarian algorithm for the linear assignment problem, including the improvements proposed by Jonker and Volgenant (1986). An extension of this algorithm can also deal with a fixed number of vehicles (see Paixão and Branco 1988). Dell'Amico (1989) (see also Dell'Amico et al. 1993) proposes an $O(n^3)$ successive shortest-path algorithm for the SDVSP, which uses the initialization proposed by Jonker and Volgenant (1986). Song and Zhou (1990) propose an $O(n^3)$ successive shortest-path algorithm as well. Bokinge and Hasselström (1980) propose a minimum-cost flow

approach that allows a significant reduction of the size of the model in terms of the number of variables at the price of an increased number of constraints.

In this paper, we consider several models and algorithms for the SDVSP. In §1, we review the quasi-assignment formulation, and consider an asymmetric assignment formulation for the SDVSP. In §2, we propose a new auction algorithm for the quasiassignment problem. In §3, we discuss a reduction technique which allows for a significant reduction of the number of variables in both assignment models. This reduction is applicable when the costs have a certain special structure. We propose a new two-phase algorithm, which is valid in the case of the special cost structure, and a new core-oriented approach that consists of iteratively solving the sparse subproblem of the original problem.

In §4, the results of an extensive computational study are presented, using both randomly generated data and real-life instances. This study includes comparisons of several algorithms proposed in this paper with the most efficient algorithms reported in the literature. The new algorithms show a significant performance improvement with respect to computation time. Such improvement is very important in the case in which the SDVSP is used as a subproblem. For example, many instances of the SDVSP have to be solved when it is used as a relaxation of more complicated scheduling problems, such as the multidepot vehicle-scheduling problem (see Dell'Amico et al. 1993, Ribeiro and Soumis 1994, Forbes et al. 1994). The research which has resulted in this paper is motivated by our research on the integration of vehicle and crew scheduling (see Freling 1997), where we need to solve thousands of instances of the SDVSP to get a solution for one instance of the original problem.

1. Mathematical Formulations

Before considering mathematical formulations, we first introduce some definitions and notation. Trips serviced by the same vehicle are linked by *deadheading trips (dh-trips)*, that is, movements of vehicles without serving passengers. *Dh-trips* consist of travel time (or *vehicle deadheading*) and/or *idle time*. Idle time is defined as the time a vehicle is idle at a location other

than the depot. Let b_i and e_i be the starting and ending locations, and let bt_i and et_i be the starting and ending times of a trip i, respectively. Two trips i and j are said to be a *compatible* pair of trips if the same vehicle can cover these trips in sequence, that is, if $et_i + trav(e_i, b_j) \leq bt_j$, where $trav(e_j, b_i)$ is the deadheading travel time from location e_j to location b_i. A sequence of trips is *feasible* if each consecutive pair of trips in the sequence is compatible.

1.1. Straightforward Formulation

Let $N = \{1, 2, \ldots, n\}$ be the set of trips, numbered according to increasing starting time, and let $E = \{(i, j) \mid i < j, i, j \text{ compatible}, i \in N, j \in N\}$ be the set of arcs corresponding to dh-trips. The nodes s and t both represent the depot at location d. We define the vehicle-scheduling network $G = (V, A)$, which is an acyclic directed network with nodes $V = N \cup \{s, t\}$, and arcs $A = E \cup (s \times N) \cup (N \times t)$. A path from s to t in the network represents a feasible schedule for one vehicle, and a complete feasible vehicle schedule is a set of disjoint paths from s to t such that each node in N is covered.

Let c_{ij} be the vehicle cost of arc $(i, j) \in A$, which is usually some function of travel and idle time. Furthermore, a fixed cost K for using a vehicle can be added to the cost of arcs (s, i) or (j, t) for all $i, j \in N$. For the remainder of this paper, we assume that the primary objective is to minimize the number of vehicles. This means that K is high enough to guarantee that this minimum number will be achieved.

Using decision variables y_{ij}, with $y_{ij} = 1$ if a vehicle covers trip j directly after trip i, $y_{ij} = 0$ otherwise, the SDVSP can be formulated as follows:

$$\min \sum_{(i, j) \in A} c_{ij} y_{ij} \qquad (1)$$

$$\sum_{j : (i, j) \in A} y_{ij} = 1 \qquad \forall i \in N, \qquad (2)$$

$$\sum_{i : (i, j) \in A} y_{ij} = 1 \qquad \forall j \in N, \qquad (3)$$

$$y_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A. \qquad (4)$$

Constraints (2) and (3) assure that each trip is assigned to exactly one predecessor and one successor, that is, these constraints guarantee that the

network is partitioned in a set of disjoint paths from s to t. Because (2) and (3) define a totally unimodular restriction matrix, binary conditions on the variables are often relaxed to $y_{ij} \geq 0$.

1.2. Assignment Formulations

Note that the model above corresponds to a linear assignment problem if nodes s and t and the corresponding arcs are not considered. Therefore, Paixão and Branco (1987) call this model a *quasiassignment* model, and the underlying problem the quasiassignment problem (QAP). The following other assignment models are possible for the SDVSP, depending on the way the depot is modeled:

1. For a linear assignment formulation (see, for instance, Paixão and Branco 1987), the depot is represented by $2n$ nodes.

2. For a two-sided inequality assignment formulation (see Freling et al. 1995), the depot is not represented at all. All costs for linking the depot with a trip are transferred to costs between trips.

3. For an asymmetric assignment formulation, the depot is represented by n nodes, corresponding to arrivals at the depot.

The choice between the models is a trade-off between the compactness of the formulation and its solvability. Because for the linear assignment formulation too much information is included, and for the two-sided inequality formulation too little information is included, we do not consider these approaches. The quasiassignment and the asymmetric assignment models are somewhere in between. The disadvantage of the asymmetric assignment model is that it includes no information about departures from the depot explicitly in the model. An advantage is that several algorithms for solving this asymmetric assignment model are proposed by Bertsekas and Castañon (1992) and a corresponding code is in public domain. Therefore, we consider this approach in more detail.

Consider network $G^T = (M, E^T)$, which is obtained from network G by deleting nodes s and t and the corresponding arcs, and introducing an artificial node $n+i$ and a zero cost arc $(i, n+i)$ corresponding to each node $i \in N$. That is, we consider the node set $M = N \cup \{n+1, \ldots, 2n\}$ and the arc set $E^T = E \cup \{(i, n+i), i = 1, \ldots, n\}$. A path of the form $\{i_1, i_2, \ldots, i_k, n+i_k\}$ corresponds to a feasible vehicle schedule leaving the depot to perform trips $i_1, i_2, \ldots, i_k$ and returning to the depot afterwards. In other words, the departure from the depot is not explicitly modeled, while the return to the depot is modeled by arcs of type $(i, n+i)$. Let

$$b_{ij} = \begin{cases} c_{ij} - c_{sj} & \forall (i, j) \in E, \\ c_{it} & \forall (i, j) \in E^T \setminus E, \end{cases}$$

that is, the costs for links with the origin depot is transferred to links between trips. With this cost definition and a similar definition of the variables, the SDVSP can be formulated as follows:

$$\min \sum_{(i, j) \in E^T} b_{ij} y_{ij} + \sum_{j \in N} c_{sj}$$

$$\sum_{j:(i, j) \in E^T} y_{ij} = 1 \qquad \forall i \in N,$$

$$\sum_{i:(i, j) \in E^T} y_{ij} \leq 1 \qquad \forall j \in M,$$

$$y_{ij} \in \{0, 1\} \qquad \forall (i, j) \in E^T.$$

2. An Auction Algorithm for the Quasiassignment Problem

In this section we propose a new algorithm for the SDVSP, that is, an auction algorithm for the quasiassignment problem. Auction algorithms are primal-dual algorithms for network flow problems. We refer to Bertsekas and Castañon (1991) for an intuitive economic interpretation of auction algorithms. Originally, auction algorithms were developed for parallel computation, but they turned out to be very fast in a sequential environment as well. In particular, auction algorithms are known to outperform other algorithms for sparse linear assignment problems. We will consider several auction algorithms in the computational study in §4. In particular, we will consider an auction algorithm for the asymmetric assignment problem proposed by Bertsekas and Castañon (1991), and a new auction algorithm for the quasiassignment problem which is the topic of this subsection.

Auction algorithms are iterative procedures, where both primal and dual costs may deteriorate at any iteration, although in the end an optimal solution

is obtained. They exist in three forms: forward, reverse, and combined forward and reverse auction algorithms. We propose a new combined forward and reverse auction algorithm for the quasiassignment problem. This algorithm is a straightforward adaptation of algorithms developed by Bertsekas (1991) for several other assignment-type problems. Therefore, we will not prove the correctness of the algorithm (the interested reader is referred to Freling 1997).

Consider an assignment of a trip or source i to a trip or sink j with $i < j$, that is, $y_{ij} = 1$. Then, we say that a trip $i \in N$ is *forward assigned* to $j \in V$, and a trip $j \in N$ is *backward assigned* to $i \in V$. A quasiassignment is feasible if each trip is forward assigned to another trip or to the sink, and backward assigned to another trip or to the source (constraints (2) and (3)). For any feasible quasiassignment Q, we distinguish between three types of assignments: trip-trip assignments $S(Q) = \{(i, j) \in E, i$ and j assigned to each other$\}$, source assignments $D_s(Q) = \{j \in N, j$ backward assigned to the source$\}$, and sink assignments $D_t(Q) = \{i \in N, i$ forward assigned to the sink$\}$. In accordance with the auction terminology, we formulate the QAP as a maximization problem by taking $a_{ij} = -c_{ij}$ for all $(i, j) \in A$. We also assume that a_{ij} is integer for all $(i, j) \in A$.

2.1. Complementary Slackness for the Quasiassignment Problem

Let π_i and p_j denote the dual variables corresponding to constraints (2) and (3), respectively. In the auction terminology π_i is the profit of a forward assignment of trip i, while p_j is the price of a backward assignment of trip j. The concept of ε-*complementary slackness* (ε-CS) is introduced in order to avoid cycling in the auction algorithm.

DEFINITION 1. A quasiassignment Q and a profit-price pair (π, p) satisfy ε-CS for the QAP if for $\varepsilon > 0$:

$$
\begin{aligned}
\pi_i + p_j &\geq a_{ij} - \varepsilon & \forall (i, j) \in E, \\
\pi_i + p_j &= a_{ij} & \forall (i, j) \in S, \\
\pi_i &\geq a_{it} & \forall i \in N, \\
\pi_i &= a_{it} & \forall i \in D_t, \\
p_j &\geq a_{sj} & \forall i \in N, \\
p_j &= a_{sj} & \forall j \in D_s.
\end{aligned} \tag{5}
$$

Suppose that a feasible quasiassignment Q together with a profit-price vector pair (π, p) satisfy ε-CS for the QAP. Moreover, define $\bar{p}_j = p_j + \varepsilon$ for all $j \in N$. Then, one can prove that Q is within $n\varepsilon$ of being an optimal quasiassignment, and the feasible dual solution given by $(\pi, \bar{p})$ is within $n\varepsilon$ of being optimal for the dual problem. Thus, to guarantee optimality of a quasiassignment solution, it is sufficient that the ε-CS condition is satisfied with $\varepsilon < 1/n$.

2.2. Combined Forward and Reverse Auction Algorithm

The combined forward and reverse auction algorithm consists of forward and backward auction iterations, where in a forward auction iteration trips are forward assigned, while in a backward auction iteration trips are backward assigned. In each forward or backward auction iteration an unassigned trip bids for another trip or the depot. For the QAP, the combined version is the only option since bidding needs to occur from trips to the sink (forwards) and from trips to the source (backwards).

The algorithm starts with an empty assignment and an arbitrary set of prices, and switches between sequences of forward and reverse auction iterations until a feasible solution is reached which can be shown to be optimal. The corresponding set of prices solves the dual problem. Intermediate solutions where not every trip is forward and backward assigned are called *partial quasiassignments*. Thus, the combined forward and reverse auction algorithm for the QAP generates a sequence of profit-price vector pairs (π, p) and partial quasiassignments, while keeping ε-CS satisfied in each iteration. The use of ε-CS as a perturbation mechanism is often necessary in order to avoid cycling. Without this mechanism it may be that several trips compete for a smaller number of equally desirable trips.

Let us consider the forward auction procedure. The value of a bid of trip i for another trip j, which is candidate for forward assignment, is denoted by $f_{ij} = a_{ij} - p_j$, while the value of a bid of trip i for the sink is denoted by $f_{it} = a_{it} + \varepsilon$, with $\varepsilon > 0$ defined as before. In a forward auction iteration, a trip is forward assigned to the candidate (trip or sink) with the maximum corresponding value. In case the maximum

value is attained for a trip, the price of this trip is raised as much as possible, while keeping ε-CS satisfied. The forward auction procedure is shown below.

Forward Auction Procedure.

Step 1. Perform the forward auction iteration in the next steps for each trip $i \in N$ which has no forward assignment.

Step 2. Determine the trip or sink j_i with the maximum bid with value $\beta_i = \max\{f_{ij} \mid j: (i, j) \in A\}$. If trip i has only one arc $(i, j) \in A$ set $\gamma_i = -\infty$; Otherwise, determine the second highest value $\gamma_i = \max\{f_{ij} \mid j: (i, j) \in A, j \neq j_i\}$. If $j_i = t$ go to Step 4.

Step 3. Update the prices: $p_{j_i} = p_{j_i} + \beta_i - \gamma_i + \varepsilon = a_{ij_i} - \gamma_i + \varepsilon$, and $\pi_i = a_{ij_i} - p_{j_i}$, and update the assignments: (i, j_i) is added to S. If trip j_i was already backward assigned, then remove the assignment from S or D_s. Return to Step 1.

Step 4. Update the price: $\pi_i = a_{it}$, and update the assignment: add i to D_t. Return to Step 1.

The reverse auction procedure is similar, with bids for candidates for forward assignments replaced by bids for candidates for backward assignments (see Freling 1997). To guarantee convergence of the algorithm, one refrains from switching between the forward and reverse auction procedures until at least one more trip becomes forward or backward assigned. Note that with this rule, at most n switches are possible until the algorithm terminates with a feasible quasiassignment. One can prove that this algorithm maintains the ε-CS conditions in Definition 1 for the pair (π, p) throughout the algorithm.

We have mentioned before that the algorithm can start with arbitrary prices and the empty assignment. Even if these initial prices do not satisfy the ε-CS conditions, they will do so after the first forward auction procedure. The reason for this is that S is empty and bids only include p_j, while the update of p_j and π_i ensures that the ε-CS conditions are satisfied. If at least one feasible quasiassignment exists, one can show that the combined forward and reverse auction algorithm terminates after a finite number of iterations, and with a feasible assignment satisfying ε-CS.

2.3. Scaling and Complexity

For good theoretical as well as practical performance, a technique called ε-scaling is used, which consists of applying the algorithm several times, starting with a large value of ε and successively reducing ε up to an ultimate value that is less than a threshold $(\frac{1}{n})$. The idea is that each scaling phase provides good initial prices for the next. Bertsekas and Eckstein (1988) show that the running time of an auction algorithm for the linear assignment problem using ε-scaling is $O(nm \log(nC))$, where n is the number of elements to assign, m is the number of possible assignments between each two elements, and C is the maximum absolute benefit. The running time for the auction algorithm for the QAP can be shown to be of the same order. The computational analysis performed by Schwartz (1994) indicates an expected computational complexity of $O(n^2 \log n)$ for a forward auction algorithm for the linear assignment problem. Furthermore, computational results in Bertsekas (1991) indicate that the combined version is considerably faster than the forward version alone.

3. Reduced Network Approaches

A bottleneck for vehicle-scheduling applications when using networks such as G or G^T is the large size of those networks, due to a large number of arcs in E. This occurs, for example, in urban bus scheduling, where the average length of a trip is relatively short compared to the whole planning horizon. Suppose we assume that when a vehicle has a long idle time between two trips it will return to the depot. Even if the vehicle does not return to the depot in practice, it is not a restrictive assumption because the pass by the depot need not be carried out. If we could simply replace the arcs corresponding to such a long idle time with the arcs for return to the depot, which already exist in network G, we could reduce the number of arcs in this network. However, in that case we cannot put a fixed cost on arcs to or from the depot, because passing by the depot should not add to the cost of using a vehicle.

In this section we propose, respectively, a two-phase approach and a core-oriented approach for the SDVSP. The two-phase approach is an efficient alternative for dealing with a large number of arcs when a special cost structure can be used. The core approach

is an efficient alternative for dealing with a large number of arcs in general. Both approaches use the quasi-assignment structure, but other assignment structures can be used as well.

3.1. Two-Phase Approach Using a Special Cost Structure

Suppose the costs in network G are defined as follows (possibly after scaling):

$$c_{ij} = \begin{cases} \min\{bt_j - et_i, trav(d, b_j) \\ \quad + trav(e_i, d)\} & \forall (i, j) \in E, \\ trav(d, b_j) & i = s, \forall j \in N, \\ trav(e_i, d) & j = t, \forall i \in N, \end{cases}$$

Furthermore, let $E^l = \{(i, j) \in E : c_{ij} = trav(d, b_j) + trav(e_i, d)\}$ and $E^s = E \backslash E^l$. We will show that under this cost definition there is no conflict between the two objectives of minimizing fleet size and minimizing the cost of operating a schedule. A similar result has appeared in an unpublished paper by Scott (1986). There, one claims that the result holds for any linear function of travel time and of idle time (i.e., the cost of an arc $(i, j) \in E^s$ is $c_{ij} = \alpha trav(e_i, b_j) + \beta(bt_j - et_i - trav(e_i, b_j))$ with $\alpha \geq 0, \beta \geq 0$), instead of a linear function of travel plus idle time (i.e., $\alpha = \beta$). However, there is a gap in the proof, and we have encountered several instances where the result does not hold when $\alpha \neq \beta$. This is an important difference since $\alpha \neq \beta$ holds for many practical applications. We next provide a two-phase algorithm which we prove to be valid in the case in which $\alpha = \beta$.

3.1.1. The Two-Phase Algorithm.
The two-phase algorithm for solving the SDVSP consists of building blocks in a first phase, and combining those blocks to a complete vehicle schedule in a second phase. Recall that a block is defined as a departure from the depot, a sequence of trips, and a return to the depot, which can be entirely assigned to one vehicle. That is, trips in a block are linked by arcs in E^s, while blocks are linked by arcs in E^l. Furthermore, define $G^s = (V, A^s)$ with $A^s = E^s \cup (s \times N) \cup (N \times t)$. A path from s to t in network G^s corresponds to one

block, while an $s - t$ path in network G corresponds to a schedule for a vehicle which may contain several blocks. Because there is no cost for idle time at the depot, all variable costs are determined by the blocks, and minimizing variable costs only is equivalent to determining a minimum cost set of disjoint $s - t$ paths in G^s such that every node is covered.

The building of blocks can be done by applying an algorithm for the quasi-assignment problem using network G^s, resulting in a set of blocks B with cost $v(B)$. Note that $v(B)$ is here defined by operational costs corresponding to short arcs only. In the second phase these blocks are combined (using arcs in E^l) to obtain a complete vehicle schedule Q with operational cost $v(Q)$ and a minimum number of vehicles which define the capital cost. We will show that this is the overall minimum number of vehicles. Note that, given a predetermined set of blocks, the number of vehicles is minimized when the number of links between blocks is maximized, because every extra link means one vehicle less. The starting time of a block is defined as the departure time from the depot, which equals the starting time of the first trip minus the travel time from the depot to the start location of the first trip. Suppose the blocks in B are sorted by increasing starting time, where ties are broken arbitrarily, and let the set of feasible combinations of blocks be defined by:

$$L = \{(i, j) : i, j \in B, i < j, (\text{ending time of } i) \\ < (\text{starting time of } j)\}.$$

The problem of combining blocks such that the minimum number of vehicles is obtained is a special case of the bipartite cardinality matching problem (see, e.g., Gerards (1997)). This problem can be solved in $O(|L|\sqrt{|B|})$ time using a maximum flow algorithm. However, because of the special structure of the problem, we propose an $O(|B|^2)$ greedy algorithm, which is easy to implement and is suitable for our purposes of combining blocks. Procedure *FIRST(i)* returns the first block in B which can be served by the same vehicle after serving i; *FIRST(i)* returns zero if no block can be served after block i. We propose the greedy

algorithm Combine below, which is initialized with $Q := B$ and $v(Q) := v(B)$.

Algorithm Combine:
while $B \neq \varnothing$ **do**
begin
    i is the first block in B;
    $B := B/\{i\}$; $j := FIRST(i)$;
    while $j \neq 0$ **do**
    begin
      $i := MERGE(i, j)$;
      $B := B/\{j\}$; $j := FIRST(j)$;
    end;
    $Q := Q \cup \{i\}$; $v(Q) := v(Q) + K$;
end;

Procedure $MERGE(i, j)$ returns the block obtained by merging i and j together, that is, one vehicle is executing block j directly after block i. The fixed cost K is added for each vehicle in use. In Freling [15], we prove that this greedy procedure will always minimize the number of vehicles given a set of blocks B. Our computation tests indicate that the running time of Algorithm Combine takes up to 1.1% of the total running time of the two-phase approach. Gupta et al. (1979) propose an $O(|B| \log |B|)$ algorithm for the minimum-resource fixed job-sheduling problem. This algorithm can be used here as well but is not worthwhile to consider since running time is not a bottleneck.

3.1.2. Validity of the Two-Phase Algorithm.

Next we will show that under the cost definition above there is no conflict between the two objectives of minimizing fleet size and minimizing the cost of operating a schedule. It is equivalent to prove that the minimum number of vehicles is always part of an optimal solution using the complete network G with operational costs and without fixed vehicle costs. After the proof we will discuss the validity of the two-phase algorithm.

PROPOSITION 2. *Using the special cost structure, the minimum number of vehicles is obtained when minimizing the operational costs only in network G.*

PROOF. Our analysis relies on the concept of augmenting paths and cycles in a residual network (see,

e.g., Ahuja et al. 1993). The residual network $G(Q)$ corresponding to a vehicle schedule Q is obtained from network G as follows: Arcs not contained in Q remain as before (forward arcs), while each arc (i, j) contained in Q is replaced by a backward arc (j, i) with cost $-c_{ij}$. A path or cycle in network $G(Q)$ is said to be alternating with respect to schedule Q if its arcs are alternately forward and backward. Recall that a solution Q consists of a set of disjoint paths in network G. An alternative solution to Q can be obtained by replacing the backward arcs with the forward arcs in an alternating path or cycle in network $G(Q)$. To remain feasible, a forward arc into a node (except for t) must be followed by a backward arc to unassign the predecessor of the node in Q, while a backward arc into a node must be followed by a forward arc to assign a new successor. Each alternating path from s to t starting and ending with a forward arc corresponds to a feasible alternative to schedule Q by adding one extra vehicle. Such a path is called augmenting. An alternating cycle is never augmenting because it does not correspond to an extra vehicle, but it corresponds to a change of Q into another feasible schedule with the same number of vehicles.

Suppose Q^{opt} is optimal when minimizing operational costs only in network G. If the proposition does not hold, then there exists a vehicle schedule Q^{aug} with fewer vehicles. The difference between Q^{opt} and Q^{aug} can be decomposed in alternating $s-t$ paths and alternating cycles in $G(Q^{opt})$. These cycles have nonnegative costs because of the definition of Q^{opt}. The existence of Q^{aug} implies that there is at least one augmenting path with nonpositive cost in $G(Q^{opt})$. Let the path AP with elements $s \to i_1 \leftarrow i_2 \to \cdots \to i_{p-1} \leftarrow i_p \to t$ be such a path, where $\to$ denotes a forward arc and $\leftarrow$ denotes a backward arc. It holds that $(i_p, i_1) \notin E^l$, because otherwise AP is an alternating cycle, which is not augmenting by definition. The proof that an augmenting path AP with nonpositive cost does not exist is built up in three steps:

1. Consider an augmenting path AP_1 with nonpositive cost, and without forward and backward arcs in E^l. With P^+ and P^- defined as the set of forward and backward arcs in AP_1, respectively, the cost of AP_1 is

given by:

$$c(AP_1) = c_{si_1} + \sum_{(i,)j\in P^+} c_{ij} - \sum_{(j,i)\in P^-} c_{ij} + c_{i_p t}$$

$$= trav(d, b_{i_1}) - (bt_{i_1} - et_{i_2}) + (bt_{i_3} - et_{i_2}) - \cdots$$

$$+ (bt_{i_{p-1}} - et_{i_{p-2}}) - (bt_{i_{p-1}} - et_{i_p}) + trav(e_{i_p}, d)$$

$$= trav(d, b_{i_1}) - bt_{i_1} + et_{i_p} + trav(e_{i_p}, d).$$

For AP_1 to have nonpositive cost, we must have

$$c(AP_1) \le 0 \Longleftrightarrow et_{i_p} + trav(e_{i_p}, d) \le bt_{i_1} - trav(d, b_{i_1}).$$

However, this implies $(i_p, i_1) \in E^l$, which is a contradiction.

2. Consider an augmenting path AP_2 with nonpositive cost, and without forward arcs in E^l. Let R^- be the set of backward arcs in $E^l \cap AP_2$ and define $z_{ij} = bt_j - et_i - trav(e_i, d) - trav(d, b_j)$ for each $(j, i) \in R^-$. We know that $z_{ij} \ge 0$ because $(i, j) \in E^l$. Similar computations as before result in

$$c(AP_2) = trav(d, b_{i_1}) - bt_{i_1} + et_{i_p} + trav(e_{i_p}, d) + \sum_{(i,j)\in R^-} z_{ij}.$$

Because $z_{ij} \ge 0$ for all $(i, j) \in E^l$ we get again that $(i_p, i_1) \in E^l$ if $c(AP_2)$ is nonpositive, which is a contradiction.

3. Consider a general augmenting path AP, which may contain any type of arc. We can split AP at each forward arc in E^l because the cost of these arcs is the sum of the corresponding travel times to t and from s. Then, we obtain a set of $s - t$ paths of the types described above, for which we have shown that each one has positive cost. Thus, AP has positive cost and an augmenting path AP with nonpositive cost does not exist.

Concluding, because an augmenting path does not exist, the corresponding vehicle schedule Q^{aug} with fewer vehicles does not exist. Therefore, Q^{opt} which is obtained by minimizing operational costs only in network G, contains the minimum number of vehicles. □

This proposition is sufficient to prove the validity of the two-phase algorithm, that is, the solution of the algorithm corresponds to a feasible vehicle schedule with a minimum number of vehicles. Recall that operational costs are minimized in the first phase, and no extra operational costs are involved in combining the blocks in the second phase. Because algorithm combine will always minimize the number of vehicles given a set of blocks (see Freling 1997), the two-phase algorithm will result in a solution with the same objective value as an optimal solution using network G with the objective of minimizing operational costs only.

3.2. Core-Oriented Approach

In general, a core-oriented approach consists of considering a reduced representation (*a core*) of a problem. The core serves as input of an algorithm, and is iteratively updated until no elements outside of the core can improve the solution. Such an approach has, for example, been successfully applied to the travelling salesman problem (Junger et al. 1995), a matching problem (see, e.g., Gerards 1997), the linear assignment problem (Volgenant 1996) and the set-covering problem (Caprara et al. 1999). We propose a core-oriented approach which enables us to consider a reduced number of arcs and thus work on sparser subproblems of the original problem. Let the core $A^c \subseteq A$ be a set of arcs in network G. We propose the following core-oriented approach:

Algorithm CoreFR

Step 1. Forward/Reverse Auction. Apply the combined forward and reverse auction algorithm of §2 to the reduced network $G^c = (V, A^c)$.

Step 2. Optimality Check. Stop if ε-CS is satisfied with $\varepsilon < \frac{1}{n}$ for the current primal and dual solution and the entire set A.

Step 3. Replace A^c. Replace the core A^c by a set which incorporates the current assignment and at least one arc which violates ε-CS. Return to Step 1 where the algorithm is initialized with the current dual values.

Any algorithm for the QAP can be used to solve the SDVSP on the reduced network in Step 1. Here, we consider the incorporation of the combined forward and reverse auction algorithm in the core approach. This enables us to exploit the well-known superior behavior of auction algorithms on sparse problems. The optimality check in Step 2 consists of checking

the first condition of (5) only, because the other conditions are already satisfied in Step 1. Note the similarity with the revised simplex method since this check corresponds to detecting positive reduced cost arcs. The essential part of the algorithm is the strategy for replacing the core in Step 3. The set A^c can be initialized and replaced in various ways but always incorporates all arcs connected with the source and the sink. We have considered several strategies (see Freling 1997), but in the computational study of §4 we consider one strategy which turned out to perform better overall on the test sets considered. For this strategy, the initial core A^c is chosen as the set of arcs in E^s, plus the k least cost arcs in E^l emanating from each trip. When selecting the k least (reduced) cost arcs, we also include all arcs not in this subset but with cost equal to the kth smallest cost arc. The core is updated in Step 3 of the algorithm by adding the least reduced cost arc emanating from each trip if the reduced cost is negative, where the reduced cost of arc $(i, j) \in A$ is defined as $\bar{c}_{ij} = c_{ij} + \pi_i + p_j + \varepsilon$. We reoptimize the quasiassignment problem by keeping those previous assignments of trips of which no new arc in E^l leaves or enters its corresponding node.

It is not easy to give good bounds on the running time of Algorithm CoreFR as a function of the input. However, suppose we use the output parameter γ, which denotes the number of times Step 1 of Algorithm CoreFR needs to be executed, and let $O(nm' \log(nC))$ denote the running time of the auction algorithm in Step 1, where m' is the maximum core size to be considered. Then, the running time of Algorithm CoreFR is $O(\gamma nm' \log(nC))$. The problem of finding the k least cost arcs among m arcs can be solved with an $O(m)$ algorithm (see Fischetti and Martello 1988). Algorithm CoreFR is also appropriate as a heuristic approach for very large instances that need a (possibly suboptimal) solution fast, where the idea is that one stops after a few updates of A^c. The core-oriented approach can also be applied to other auction or network flow algorithms.

An alternative approach for dealing with a large number of arcs is to use a minimum-cost flow formulation proposed by Bokinge and Hasselström (1980) (see also Desrosiers et al. 1995), where it is not necessary to consider arcs in E^l explicitly. The network underlying this approach uses a range of depot nodes to allow passes by the depot, while the fixed cost is imposed on one arc combining the last with the first depot node. The flow in this arc equals the number of vehicles in the solution. Any minimum cost flow algorithm can be used to solve the resulting problem.

4. Application to Bus Scheduling
We will compare the performance of the algorithms for the SDVSP proposed in this paper with the most efficient algorithms known from literature. To be able to draw general conclusions we have used both real-life and randomly generated data which involve different settings of bus scheduling. The following algorithms, implemented in the C programming language, are either new or used for the first time for the SDVSP (n and m denote the number of nodes and arcs in the corresponding networks):

- QAFR: The $O(nm \log(nC))$ combined forward and reverse auction algorithm for the quasiassignment problem proposed in §2.
- ASFR: An $O(nm \log(nC))$ forward and reverse auction algorithm for the asymmetric assignment problem, similar to algorithm asfr2 proposed in Bertsekas and Castañon (1992).
- Core: The $O(\gamma nm' \log(nC))$ Algorithm CoreFR proposed in §3.2.

The following algorithms are frequently applied to the SDVSP:

- Hung: The $O(n^3)$ Hungarian algorithm for the quasiassignment problem proposed by Paixão and Branco (1987). The code we have used is a C translation of the FORTRAN implementation by Branco (1989).
- SSP: An $O(n^3)$ successive shortest-path algorithm for the quasiassignment problem proposed by Dell'Amico (1989). We used the original executable provided by the author, which is generated from a FORTRAN code.
- MICO: The minimum cost flow approach proposed by Bokinge and Hasselström (1980). The minimum cost flow code is a C translation (line-by-line) of the public domain FORTRAN implementation RELAX4 of an $O(n^3 \log(nC))$ relaxation algorithm by Bertsekas and Tseng (1991).

The two-phase procedure for the special cost structure, using QAFR in the first phase and Algorithm

Combine in the second phase, will be denoted by QAFR+. For comparison we have also tested the two-phase procedure with algorithms ASFR, Hung, and SSP for the first phase, denoted by ASFR+, Hung+, and SSP+. All codes were run on a Pentium 400Mhz/64MB personal computer (32-bit environment). Except for SSP, all codes were generated with the MS Visual C\C++ 5.0 compiler. The SSP code was provided by Dell'Amico, who used a Watcom C/C++ compiler. Random numbers are generated using the built-in pseudorandom number generator provided with the MS Visual C/C++ compiler. We assume all data to be integer, since for the algorithms that assume integer costs we can deal with rational costs by scaling. As mentioned before, the practical performance of auction algorithms is often considerably improved by using ε-scaling, which consists of applying the algorithm several times, starting with a large value of ε and successively reducing ε to a value less than $1/n$. Each application of the algorithm, called the scaling phase, provides good initial prices for the next phase. To work with integer ε, we scale the cost: c_{ij} is multiplied by n for all $(i, j) \in A$. Then, optimality is assured when the algorithm converges with $\varepsilon < 1$. A consequence of this rescaling is that integers may get out of range, but Bertsekas (1992) notes that in case of integer overflow, it is possible to use floating point arithmetic for price update calculations, while using integer arithmetic for all other calculations. The sequence of ε values used, as proposed in Bertsekas (1991) for the linear assignment problem, is determined according to the rule $\varepsilon_i = \max\{1, \varepsilon_{i-1}/\theta\}$, in each scaling phase $i > 1$, where $\theta > 1$. The values we have used for these parameters are obtained as a result of fine tuning on one class of data (Class A to be defined later) and for the sake of comparison have been fixed for other classes of data. For QAFR and ASFR we used the values $\varepsilon_0 = nC$ and $\theta = 4$, and the adaptive form of ε-scaling (see Bertsekas 1991) whereby, within the ith scaling phase, the value of ε is gradually increased to the value ε_i given above, starting from a relatively small value. For the Core algorithm we used $\varepsilon_0 = nC/10$, $\theta = 4$, and $k = 30$. We have tested the Core algorithm for several variations of parameter k up to $k = 150$, but runtimes did not differ significantly.

4.1. Computational Experiments with Randomly Generated Data

The random data has been generated as suggested in Dell'Amico et al. (1993), to simulate real-life public transport instances in which there are short and long trips running from 5 a.m. to midnight. Long trips correspond to extra urban journeys, or to sequences of urban journeys. Short trips correspond to urban journeys and are generated so as to determine peak hours around 7–8 a.m. and 5–6 p.m. We consider five classes of problems, denoted by A, B, C, D, and E. Classes A, D, and E contain 40% short trips and 60% long trips as in Dell'Amico et al. (1993), while Class B contains short trips only, and Class C long trips only. For all classes, the costs in terms of the quasi-assignment network are defined as:

1. $c_{ij} = \alpha trav(e_i, b_j) + \beta(bt_j - et_i - trav(e_i, b_j))$ for each arc $(i, j) \in E^s$, and $c_{ij} = \alpha(trav(d, b_j) + trav(e_i, d))$ for each arc $(i, j) \in E^l$.

2. $c_{si} = \alpha trav(d, b_i) + K/2$ for all trips i, and $c_{jt} = \alpha trav(e_j, d) + K/2$ for all trips j.

For Classes A, B, C, and E, the fixed cost K for using a vehicle is equal to 2,000, while this value is 5,000 for Class D. For Classes A, B, C, and D, $\alpha = 10$ and $\beta = 2$, so that the special cost structure defined in §3.1 does not apply here, while $\alpha = \beta = 1$ for Class E where the special structure does apply.

Table 1 compares the performances of algorithms Hung, SSP, MICO, QAFR, ASFR, and Core for instances of Class A. For each value of the number of trips n, varying from 100 to 1,500, 10 instances of the SDVSP were generated. The first three columns in Table 1 give the value of n, the optimal cost, and the corresponding number of vehicles, respectively, where the last two numbers are rounded-off averages. The remaining columns show the average (and, below that, the maximum) computing times in seconds, excluding input and output, for the six algorithms considered. The code corresponding to algorithm SSP has a limit of 999 trips, so no results are presented for $n \geq 1000$.

The table shows that the auction algorithms Core and QAFR outperform the other algorithms for these instances. Algorithm Core has the fastest average computing time for nine of 15 cases, which shows that it is worthwhile to consider a core approach for these

Table 1   Average and Maximum Run Times for Class A Instances

Trips	Cost	Vehicles	Hung.	SSP	MICO	QAFR	ASFR	Core
100	88,991	29	0.02	0.06	0.04	0.02	0.01	0.04
			0.16	0.16	0.17	0.11	0.11	0.17
200	165,749	55	0.44	0.32	0.73	0.08	0.04	0.13
			0.66	0.49	1.21	0.38	0.17	0.55
300	238,595	79	1.39	0.83	2.54	0.33	0.33	0.29
			1.81	1.21	5.17	0.98	0.77	0.83
400	305,545	102	3.64	1.58	6.04	0.70	0.74	0.82
			5.06	2.30	16.36	2.03	1.54	5.99
500	374,666	125	7.48	2.79	13.26	1.13	1.57	1.06
			9.01	4.23	32.79	3.24	2.74	5.93
600	446,478	149	14.09	4.61	31.57	2.17	3.03	2.47
			16.86	6.59	111.00	9.07	7.69	22.14
700	515,486	176	20.00	7.22	47.95	3.06	4.12	1.79
			23.73	10.00	108.10	9.01	9.56	7.85
800	588,346	200	35.80	10.32	80.24	4.06	6.14	3.24
			40.87	13.46	191.96	13.96	23.89	16.97
900	662,635	222	42.22	13.53	132.58	5.51	10.90	3.78
			47.79	17.46	340.38	11.37	31.80	13.29
1,000	712,364	244	60.73	**	182.34	8.30	11.92	8.73
			71.90	**	507.63	22.63	28.29	120.67
1,100	786,448	268	79.27	**	289.67	11.92	21.07	11.03
			97.44	**	703.27	22.52	88.70	103.15
1,200	846,358	291	90.00	**	282.02	12.96	18.75	9.77
			120.83	**	767.53	50.86	89.09	125.89
1,300	909,317	317	113.19	**	381.74	15.29	24.98	20.97
			141.88	**	912.04	35.27	70.09	230.20
1,400	974,579	336	130.21	**	433.69	22.57	71.95	16.55
			145.22	**	1110.76	49.71	1055.94	151.93
1,500	1,051,741	363	134.02	**	611.40	20.80	34.36	12.29
			180.71	**	1623.27	75.91	133.69	67.94

type of problems. However, the algorithm seems to be less stable compared to, for example, QAFR. This is confirmed by our other test results reported later in this section. Algorithm Hung seems most stable because the maxima are relatively close to averages. An explanation might be that this algorithm does not use scaling. Corresponding to the tests reported in Table 1, we show in Table 2 the number of nodes and the density of network G and the minimum cost flow network, and the density of the core approaches. The density of network G is defined by $m/(n+1)^2 * 100\%$, and is around 30% for these problems. For Core these densities correspond to the density of the final core,

when the algorithm has converged to an optimal solution.

Note the increase in the number of nodes and the decrease in the density for the minimum cost flow network compared to network G. Also note the relatively small sizes of the core needed to detect optimality, while the density decreases with the size of the instances.

Table 3 reports the average computation times for Classes B, C, and D, in seconds over ten instances for different values of n. For Class B (with short trips only), algorithm Core shows an increase in running times of one order of magnitude compared to the

Table 2   Network Sizes for Class A Instances

QAFR Nodes	MICO Nodes	QAFR Density	MICO Density	Core Density
102	241	29.3	15.4	23.5
202	477	29.6	13.5	19.6
302	716	29.9	13.3	18.2
402	954	30.0	13.0	16.8
502	1,186	29.9	13.4	17.4
602	1,419	29.8	13.3	16.8
702	1,653	29.6	13.1	16.3
802	1,883	29.7	13.0	15.8
902	2,112	29.7	13.1	15.6
1,002	2,346	29.9	12.9	15.3
1,102	2,570	29.9	13.0	15.7
1,202	2,796	29.9	13.0	15.6
1,302	3,028	30.0	12.9	15.3
1,402	3,247	30.0	12.8	15.0
1,502	3,467	30.0	12.8	15.0

results in Table 1. The other algorithms show a similar relative performance compared to Table 1. This is also the case for the results of Classes C and D, except that the average running times for ASFR are very large for the instances with $n = 1,500$. This is due to one outlier, which shows that ASFR is also not stable under all conditions.

Table 4 compares the performance of algorithms Hung+, SSP+, QAFR+, ASFR+, Hung, SSP, MICO, QAFR, ASFR, and Core for instances of Class E. The density of network G is around 35%, while the density of network G^s is around 3% for these problems. The average and maximum computing times are shown in the table for 10 instances per value for n.

The table shows that the two-phase algorithms Hung+, SSP+, QAFR+, and ASFR+, exploiting the special cost structure, perform considerably better than the other algorithms, both with respect to average and maximum running times. Among these algorithms QAFR+ performs best. When not using the reduced network by exploiting the special cost structure, algorithms MICO and SSP outperform the auction algorithms for the instances up to $n = 900$, as opposed to the results for the Class A instances in Table 1. This difference in relative performance of the auction algorithms may be explained by the difference in cost structure. Because operational costs are lower, there is an increased possibility for price wars,

where two trips have equal bid value for one other trip. We do not have an explanation for the large difference in average and maximum computing times when comparing MICO for Class A and Class E instances. Note that here the results of Core are worse compared to QAFR.

4.2.   Computational Experiments with Real-Life Data

In addition to these randomly generated problems, we have tested algorithms on two data sets from real-life cases. These instances are, relatively, very easy to solve, which can be seen from the running times reported in this section. The first set consists of instances from the CARRIS public transport company of Lisbon, Portugal, which correspond to clusters of bus lines covering peak hours only. Instead of considering one or a few lines for the entire day, one prefers to consider a lot of lines in parts of the day, and obtain the line schedules by combining the resulting vehicle schedules. We have, somewhat arbitrarily, used $\alpha = 2$, $\beta = 1$, and $K = 2,000$ for the cost definitions. The special cost structure does not apply, but the division between arcs in E^s and E^l is not used here anyway because of the short planning horizon. For this reason, we do not consider the core approach. Table 5 gives the performance of algorithms Hung, SSP, MICO, QAFR, and ASFR. For each value of the number of trips n, the table shows the density of G, the cost of the optimal solution, and the corresponding number of vehicles. The remaining columns show the computing times in seconds, excluding input and output, for the five algorithms considered. The table shows that algorithms HUNG, QAFR, and ASFR are extremely fast for these instances, although all algorithms run fast due to the low density of the networks.

In addition, Table 6 compares the performance of algorithms Hung+, SSP+, QAFR+, ASFR+, Hung, SSP, MICO, QAFR, ASFR, and Core, for instances from the RET public transport company of Rotterdam, the Netherlands. These instances correspond to bus lines or clusters of bus lines covering one day, and satisfy the special cost structure. The primary objective is to minimize the number of buses, while the secondary is to minimize the total time a bus is out of the depot. The cost definitions are equivalent to those of Class E,

Table 3   Average and Maximum Run Times for Class B, C, and D Instances

	Trips	Cost	Vehicles	Hung	SSP	MICO	QAFR	ASFR	Core
	100	67,215	19	0.04	0.07	0.04	0.01	0.01	0.05
				0.11	0.16	0.16	0.11	0.06	0.22
	300	186,051	56	1.98	1.11	2.46	0.32	0.57	0.47
				2.58	1.48	5.00	0.44	0.98	1.65
	500	292,989	90	10.48	4.70	12.89	1.20	2.43	3.58
				12.80	7.25	25.33	1.82	3.63	18.56
	700	406,136	127	26.12	12.53	40.78	3.21	5.65	21.28
				29.17	18.23	94.31	4.78	11.58	67.28
Class B	900	503,310	159	60.04	25.69	80.39	6.64	16.39	96.45
				70.30	35.26	148.79	13.24	90.02	189.22
	1,100	616,789	196	101.39	**	174.73	12.58	23.01	213.36
				125.62	**	362.94	19.72	48.34	632.96
	1,300	728,852	233	148.90	**	283.66	22.19	35.16	603.53
				167.75	**	569.25	40.10	74.14	1,363.53
	1,500	835,609	268	220.74	**	472.03	51.55	88.01	1,309.31
				236.51	**	810.76	135.61	212.40	2,058.66
	100	111,485	38	0.02	0.04	0.06	0.01	0.00	0.02
				0.11	0.16	0.66	0.05	0.00	0.27
	300	292,303	100	0.99	0.62	1.92	0.20	0.24	0.21
				1.26	0.99	5.38	0.66	0.55	0.55
	500	469,262	160	4.84	2.26	8.96	0.74	1.34	0.67
				5.71	3.90	19.39	2.08	4.83	3.02
	700	632,173	222	13.98	5.53	34.77	1.58	8.08	2.14
Class C				16.64	8.12	93.43	4.29	107.43	25.32
	900	800,359	283	27.25	9.63	67.55	3.91	4.85	2.20
				30.37	14.83	354.82	13.74	13.95	4.89
	1,100	969,804	339	53.75	**	143.36	5.65	8.83	3.79
				66.46	**	465.77	9.83	13.68	10.11
	1,300	1,141,944	397	81.32	**	231.97	10.39	17.02	9.80
				98.38	**	446.21	22.85	40.10	40.75
	1,500	1,289,315	456	115.39	**	325.53	21.99	216.41	29.50
				136.93	**	943.24	78.87	3,724.17	179.22
	100	333,732	30	0.07	0.07	0.11	0.04	0.01	0.07
				0.22	0.17	0.55	0.22	0.06	0.27
	300	882,398	80	1.33	0.74	2.72	0.47	0.41	0.46
				1.54	1.16	10.33	1.86	0.66	1.87
	500	1,372,185	124	6.25	2.57	12.28	1.42	1.88	1.04
				7.14	3.41	21.70	2.69	3.02	2.31
	700	1,906,643	173	17.85	7.23	48.71	3.58	5.82	2.31
Class D				20.65	9.94	106.67	10.11	19.34	7.36
	900	2,392,535	218	36.77	12.74	113.51	7.25	11.55	6.27
				40.59	17.85	283.08	15.16	20.65	23.02
	1,100	2,928,803	268	66.22	**	286.91	9.82	18.77	12.47
				77.44	**	929.23	17.47	31.31	65.03
	1,300	3,460,637	317	106.22	**	472.83	17.65	28.18	10.95
				124.41	**	1,414.33	38.83	47.35	36.14
	1,500	3,969,610	365	140.98	**	779.77	23.90	55.30	10.92
				158.24	**	3,280.26	55.31	100.56	24.60

Table 4   Average and Maximum Run Times for Class E Instances

Trips	Cost	Veh.	Hung+	SSP+	QAFR+	ASFR+	Hung	SSP	MICO	QAFR	ASFR	Core
100	62,744	29	0.01	0.00	0.01	0.01	0.01	0.00	0.01	0.00	0.00	0.01
			0.06	0.05	0.06	0.06	0.06	0.05	0.06	0.06	0.00	0.06
300	168,784	79	0.04	0.03	0.01	0.01	0.17	0.11	0.08	0.12	0.09	0.41
			0.06	0.06	0.06	0.06	0.27	0.22	0.17	0.60	0.16	1.49
500	269,432	126	0.14	0.13	0.04	0.09	0.80	0.41	0.20	0.62	0.74	1.36
			0.33	0.22	0.12	0.33	0.99	0.50	0.33	3.29	2.64	3.62
700	365,839	172	0.67	0.66	0.18	0.27	5.78	2.95	1.21	3.27	6.86	11.53
			1.81	1.43	0.56	0.98	11.21	6.26	2.42	9.67	40.59	32.46
900	466,257	220	0.99	1.22	0.21	0.43	11.06	5.94	2.12	6.35	9.32	27.49
			2.53	3.30	0.50	1.98	22.24	13.40	7.08	22.68	27.24	106.39
1,100	563,984	266	0.64	**	0.15	0.80	7.98	**	2.01	3.88	7.01	14.05
			1.10	**	0.33	5.44	10.10	**	4.83	25.33	24.11	36.20
1,300	663,514	313	1.11	**	0.25	3.52	12.62	**	3.45	5.52	17.33	22.49
			1.70	**	0.49	22.74	15.00	**	7.09	9.50	147.48	73.27
1,500	774,283	367	1.45	**	0.34	3.10	17.02	**	4.83	8.50	17.37	43.97
			2.69	**	0.93	24.61	18.12	**	7.20	26.53	50.31	227.50

Table 5   Run Times for CARRIS Instances

Trips	Cost	Vehicles	Hung	SSP	MICO	QAFR	ASFR
255	260,250	64	0.00	0.00	6.76	0.00	0.00
255	260,250	64	0.05	0.05	6.70	0.00	0.00
255	260,729	64	0.00	0.17	6.81	0.00	0.00
255	260,254	64	0.00	0.23	6.70	0.00	0.00
255	260,729	64	0.06	0.34	6.71	0.00	0.00
255	260,291	64	0.06	0.39	6.65	0.05	0.00
255	260,732	64	0.00	0.50	6.70	0.00	0.00
255	264,370	65	0.00	0.55	6.65	0.00	0.00
255	264,762	65	0.00	0.66	6.64	0.05	0.06
352	263,477	65	0.00	0.83	6.65	0.00	0.00
352	263,487	65	0.00	0.94	6.70	0.00	0.00
352	275,653	68	0.00	0.99	6.75	0.00	0.00
192	199,069	49	0.00	1.05	6.65	0.06	0.00
192	199,435	49	0.05	1.11	6.71	0.06	0.00
192	199,069	49	0.00	1.16	6.59	0.00	0.00
192	199,435	49	0.00	1.16	6.65	0.00	0.00
192	199,125	49	0.00	1.22	6.81	0.06	0.00
192	199,439	49	0.00	1.28	6.75	0.00	0.00
192	199,234	49	0.00	1.33	6.70	0.00	0.00
192	199,482	49	0.00	1.38	6.70	0.00	0.00
264	198,639	49	0.05	1.43	6.76	0.00	0.00
264	198,647	49	0.00	1.43	6.70	0.00	0.00
264	202,703	50	0.00	1.43	6.82	0.00	0.00

and the presentation is the same as in Table 4. The density of network G^s varies between 0.22 and 3.05, while the density of network G varies between 43.77 and 47.35.

The table shows that QAFR+ and ASFR+ are extremely fast; the problem size seems to have no effect on the performance. The reason for this is probably the sparsity of the network but also the structure, which corresponds to several bus lines, while on each line, departure and arrival times are matched to allow for an efficient bus schedule. When comparing the performance of the denser problems, we can see that algorithm MICO is the fastest algorithm, similar to the results in Table 4.

5.   Conclusions

As an addition to the existing algorithms for the SDVSP, we have proposed four new algorithms: an existing auction algorithm for the asymmetric assignment problem, a new auction algorithm for the quasiassignment problem, a two-phase approach consisting of determining vehicle blocks first and combining them afterwards, and, finally, a core-oriented approach. We have shown that the two-phase approach is valid under certain cost assumptions. The choice of auction algorithms for testing the two-phase and the core approach was motivated by their known computational efficiency for network flow problems in general, and linear assignment problems in particular. Our computational results confirm that this is also true for the auction algorithms proposed in this paper

Table 6   Run Times for RET Instances

Trips	Cost	Veh.	Hung+	SSP+	QAFR+	ASFR+	Hung	SSP	MICO	QAFR	ASFR	Core
148	23,487	11	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.06	0.00	0.00
248	21,599	10	0.00	0.06	0.00	0.00	0.06	0.17	0.00	0.33	0.06	0.22
238	23,452	11	0.05	0.05	0.00	0.00	0.11	0.17	0.00	0.22	0.00	0.33
113	17,486	8	0.06	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.00
171	13,145	6	0.00	0.06	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00
197	21,278	10	0.00	0.05	0.00	0.00	0.06	0.11	0.00	0.05	0.00	0.06
72	11,351	5	0.00	0.00	0.05	0.06	0.00	0.00	0.00	0.00	0.00	0.00
126	9,016	4	0.06	0.00	0.05	0.06	0.00	0.00	0.00	0.00	0.00	0.00
127	10,988	5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
386	46,696	22	0.06	0.16	0.06	0.00	0.32	0.66	0.05	0.99	0.28	1.64
486	45,051	21	0.06	0.22	0.00	0.00	0.82	0.71	0.00	0.71	0.27	2.47
634	68,294	32	0.11	0.33	0.00	0.00	1.87	1.75	0.10	1.54	0.49	8.95
949	113,067	53	0.28	0.77	0.06	0.00	4.72	6.92	0.11	3.57	1.53	3.68
1,096	136,417	64	0.28	**	0.00	0.00	6.26	**	0.06	5.00	2.26	5.17
1,328	159,855	75	0.33	**	0.00	0.00	10.82	**	0.11	7.30	3.41	7.74

for the SDVSP. The computational study consists of randomly generated data as well as real-life data from two European public transport companies. We have provided a comparison, in terms of computational speed, of the algorithms proposed in this paper, and the most efficient algorithms for the SDVSP known from the literature.

For the majority of the test cases considered, the algorithms proposed in this paper outperform the algorithms known from the literature. When considering all test classes, the new auction algorithm for the quasiassignment algorithm (QAFR) appears to be the fastest and most stable algorithm, on average. The other auction algorithms (ASFR and Core) show good results, but are less stable for some problem classes. The computational results also show that a large reduction in computing time is obtained when exploiting the special cost structure. For the Class A, C, and D instances, the core approach performs best, while this approach shows poor results for the Class B and E instances. When not using the reduced networks for the Class E and the RET instances, the successive shortest-path algorithm and the minimum cost flow algorithm are good alternatives, although the larger instances could not be tested for the first algorithm, and the second algorithm shows very poor results on the Class A problems. It is an interesting topic for future research to detect the reason for this poor performance and to try another code for solving the minimum cost flow problems.

References

Ahuja, R. K., T. L. Magnanti, J. B. Orlin. 1993. *Network Flows, Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ.

Bertsekas, D. P. 1991. *Linear Network Optimization: Algorithms and Codes*. MIT Press, Cambridge, MA.

———. 1992. Auction algorithms for network flow problems: A tutorial introduction. *Comput. Optim. Appl.* **1** 7–66.

———, D. A. Castañon. 1992. A forward/reverse auction algorithm for asymmetric assignment problems. *Comput. Optim. Appl.* **1** 277–297.

———, J. Eckstein. 1988. Dual coordinate step methods for linear network flow problems. *Math. Programming* **42** 203–243.

Bokinge, U., D. Hasselström. 1980. Improved vehicle scheduling in public transport through systematic changes in the time-table. *Euro. J. Oper. Res.* **5** 388–395.

Branco, I. 1989. *Algoritmos para modelos matemáticos de quasi-afectacão e extensões*. Ph.D. thesis, FCUL, Universidade de Lisboa, Lisbon, Portugal.

Caprara, A., M. Fischetti, P. Toth. 1999. A heuristic algorithm for the set covering problem. *Oper. Res.* **47** 730–743.

Daduna, J. R., J. M. Pinto Paixão. 1995. Vehicle scheduling for public mass transit—An overview. J. R. Daduna, I. Branco, J. M. Pinto Paixão, eds. *Computer-Aided Transit Scheduling: Proceedings of the Sixth International Workshop*. Springer Verlag, Berlin, Germany, 76–90.

Dell'Amico, M. 1989. Una nuova procedura di assegnamento per il vehicle scheduling problem. *Ricerca Oper.* **5** 13–21.

——, M. Fischetti, P. Toth. 1993. Heuristic algorithms for the multiple depot vehicle scheduling problem. *Management Sci.* **39** 115–125.

Desrosiers, J., Y. Dumas, M. M. Solomon, F. Soumis. 1995. Time constrained routing and scheduling. M. O. Ball, T. L. Magnanti, C. L. Monma, G. L. Nemhauser, eds. *Network Routing*, Vol. 8, *Handbooks in Operations Research and Management Science*. North-Holland, Amsterdam, The Netherlands, 35–139.

Fischetti, M., S. Martello. 1988. A hybrid algorithm for finding the kth smallest of n elements in $O(n)$ time. *Ann. Oper. Res.* **13** 401–419.

Forbes, M. A., J. N. Holt, A. M. Watts. 1994. An exact algorithm for multiple depot bus scheduling. *Euro. J. Oper. Res.* **72** 115–124.

Freling, R. 1997. Models and techniques for integrating Vehicle and Crew Scheduling. Ph.D. thesis, Tinbergen Institute, Erasmus University, Rotterdam, The Netherlands.

——, J. M. Pinto Paixão, A. P. M. Wagelmans. 1995. Models and algorithms for vehicle scheduling. Working paper, Technical Report 9562/A, Econometric Institute, Erasmus University Rotterdam, Rotterdam, The Netherlands.

Gerards, A. M. H. 1997. Matching. M. O. Ball, T. L. Magnanti, C. L. Monma, G. L. Nemhauser, eds. *Network Models*, Vol. 7, *Handbooks in Operations Research and Management Science*. North Holland, Amsterdam, The Netherlands, 135–224.

Gupta, U. I., D. T. Lee, J. Y. T. Leung. 1979. An optimal solution for the channel-assignment problem. *IEEE Trans. Computers* **C-28** 807–810.

Jonker, R., T. Volgenant. 1986. Improving the Hungarian assignment algorithm. *Oper. Res. Lett.* **5** 171–176.

Junger, M., G. Reinelt, G. Rinaldi. 1995. The traveling salesman problem. M. O. Ball, T. L. Magnanti, C. L. Monma, G. L. Nemhauser, eds. *Network Models*, Vol. 7, *Handbooks in Operations Research and Management Science*. North Holland, Amsterdam, The Netherlands, 225–330.

Paixão, J. M., I. Branco. 1987. A quasi-assignment algorithm for bus scheduling. *Networks* **17** 249–269.

——, ——. 1988. Bus scheduling with a fixed number of vehicles. J. R. Daduna, A. Wren, eds. *Computer-Aided Transit Scheduling: Proceedings of the Fourth International Workshop*. Springer Verlag, Berlin, Germany, 28–40.

Ribeiro, C. C., F. Soumis. 1994. A column generation approach to the multiple-depot vehicle scheduling problem. *Oper. Res.* **42** 41–52.

Schwartz, B. L. 1994. A computational analysis of the auction algorithm. *Euro. J. Oper. Res.* **74** 161–169.

Scott, D. 1986. Minimal fleet size in transhipment-type vehicle scheduling problems. Technical Report 487, Université de Montréal, Montréal, Canada.

Song, T., L. Zhou. 1990. A new algorithm for the quasi-assignment problem. *Ann. Oper. Res.* **24** 205–223.

Volgenant, A. 1996. Linear and semi-assignment problems. *Comput. Oper. Res.* **23** 917–932.