Models and Algorithms for Vehicle Scheduling

Richard Freling[1]
Tinbergen Institute, Faculty of Economics
Erasmus University Rotterdam, The Netherlands

José M. Pinto Paixão
DEIO, Faculdade de Ciências
Universidade de Lisboa, Portugal

Albert P.M. Wagelmans
Econometric Institute, Faculty of Economics
Erasmus University Rotterdam, The Netherlands

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 the polynomially solvable case in which there is a single depot and one type of vehicle. Several assignment types of formulations are discussed and a new quasi-assignment algorithm is provided. This algorithm is based on an auction algorithm for the linear assignment problem. Furthermore, several ways for dealing with a large number of feasible connections between trips, a bottleneck for vehicle scheduling, are discussed. An important contribution of this paper is a computational study that compares our algorithms with the most successful algorithms for the vehicle scheduling problem, using both randomly generated and real life data. This can serve as a guide for both scientists and practioners. The new quasi-assignment algorithm is shown to be considerably faster for most of the test problems. This improvement in computation time is also 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 at location d and n trips from locations b_i to e_i, with corresponding fixed times bt_i and et_i (i=1,...,n), and given the travelling times between all pairs (d,b_i), (b_i,e_i), (e_i,b_j) and (e_i,d), find a feasible *minimum cost* schedule for the vehicles, such that all trips are covered by a vehicle. Each trip has to be entirely serviced by one vehicle and trips serviced by the same vehicle are linked by *deadheading trips* (*dh*-trips). These are trips without serving passengers (pairs (d,b_i), (e_i,b_j) and (e_i,d)), consisting of travel time (vehicle deadheading) and/or *idle time* (vehicle waiting time). A schedule for a vehicle is composed of vehicle *blocks*, where each block is a departure from the depot, the service of a sequence of trips and the return to the depot. The cost function is a combination of vehicle capital (fixed) and/or operational (variable) cost. The capital cost is often such that the number of vehicles will be minimized,

[1]corresponding author, email: freling@opres.few.eur.nl

while the operational cost is often a combination of vehicle travel and idle time. The SDVSP as presented here is polynomially solvable and has been formulated as a transportation model, an assignment model, a minimum cost flow model, a quasi-assignment model and a matching model.

The assignment model is obtained by considering a node pair for each trip and n node pairs for the depot. The quasi-assignment model differs from an assignment model, because the n node pairs for the depot are replaced by one node pair to which multiple assignments are allowed. This approach has been suggested by Paixão and Branco [14], who have shown that their quasi-assignment approach of the SDVSP outperforms the transportation and pure assignment approaches, so we will not consider those approaches in this paper. Their quasi-assignment algorithm is a modified version of the Hungarian algorithm for assignment problems including the improvements proposed by Jonker and Volgenant [13], and has a worst-case complexity of $O(n^3)$. An extension of this algorithm can also deal with a fixed number of vehicles (see Paixão and Branco [15]). These algorithms are applied to bus scheduling at a large bus operator in Portugal. Song and Zhou [19] propose another $O(n^3)$ quasi-assignment algorithm based on the successive shortest path algorithm for the assignment problem. They discuss applications to bus and tourist guide scheduling in China. Overviews of algorithms and applications for the SDVSP and some of its variations can be found in Daduna and Paixão [8] and in Desrosiers et al. [10].

In this paper, we compare several models and algorithms for the SDVSP. In the next section, we review the quasi-assignment formulation and show that the SDVSP can also be formulated as an inequality constrained assignment problem and as an asymmetric assignment problem. We propose a new algorithm for the quasi-assignment problem which is a modification of an auction algorithm for the linear assignment problem (Section 2). For this type of problem, auction algorithms have shown to be computational superior to Hungarian and successive shortest path algorithms. Our computational results indicate that this is also the case for the SDVSP. An improvement in the computation time can be very important when the SDVSP needs to be solved frequently for medium to large sized instances. This occurs, for example, when the SDVSP appears as relaxation of other more complicated scheduling problems, such as the multi-depot vehicle scheduling problem (see, e.g., Ribeiro and Soumis [16]), or the integrated vehicle and crew scheduling problem (see Freling et al. [11]). In Section 3, we discuss several ways for dealing with a large number of feasible links between trips. In a planning horizon of one day, this occurs for example when morning trips can be linked with most of the trips in the remainder of the day. The results of an extensive computational study, using both randomly generated data and real life instances, are presented in Section 4. This study includes comparisons of several algorithms proposed in this paper with the most efficient algorithms reported in the literature.

1. ASSIGNMENT FORMULATIONS

WE CONSIDER ASSIGNMENT type formulations for the SDVSP, and show their equivalence. We say that two trips i and j are *compatible* if $trav(e_i,b_j) \leq bt_j - et_i$, where $trav(e_j,b_i)$ is the deadheading travel time from location e_j to location b_i. Let N be the set of trips and E the set of *dh*-trip arcs (e_i,b_j) between compatible trips i and j; the nodes s and t represent the 'departure' and 'arrival' depot, both at the same location d. We define the quasi-assignment or vehicle scheduling network $G=(V,A)$, which is an acyclic directed network with

nodes $V=N\cup\{s,t\}$, and arcs $A=E\cup(s\times N)\cup(N\times t)$ corresponding to the set of *dh*-trips (e_i,b_j) between compatible trips, and to *dh*-trips (s,b_i) and (e_i,t) between each trip and the depot. Nodes can be ordered by increasing starting time so that only arcs (i,j) between compatible trips i and j with $i<j$ exist. 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 (trip) is covered. Let c_{ij} be the vehicle cost of arc $(i,j)\in A$, which is usually some function of travel and waiting time. Furthermore, let $c_{jt}=c_{jt}+K$, where K is the fixed cost for using a vehicle in the solution. 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 the following quasi-assignment model (Paixão and Branco [14]):

Quasi-Assignment Model

$$\text{minimize} \quad \sum_{(i,j)\in A} c_{ij}\, y_{ij}$$

subject to

$$\sum_{j:(i,j)\in A} y_{ij} = 1 \qquad \forall\, i\in N \qquad (2.1a)$$

$$\sum_{i:(i,j)\in A} y_{ij} = 1 \qquad \forall\, j\in N \qquad (2.1b)$$

$$y_{ij} \in \{0,1\} \qquad \forall\, (i,j)\in A \qquad (2.1c)$$

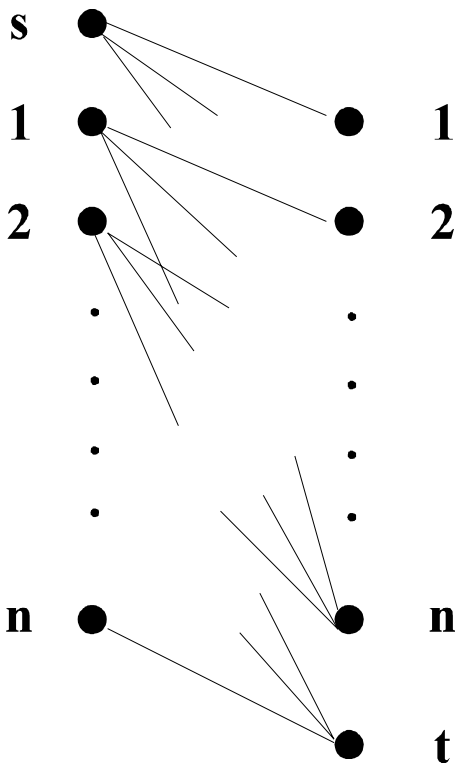

Figure 1-*Bipartite Representation*

Constraints (2.1a) and (2.1b) assure flow feasibility and the assignment of each trip to exactly one vehicle. The integer condition (2.1c) may as well be relaxed to $y_{ij}\geq 0$, for all $(i,j)\in A$, because the coefficient matrix is totally unimodular. Figure 1 shows an alternative representation of the SDVSP as a bipartite graph which illustrates the quasi-assignment structure of the problem. On the left side we consider the departure depot (s) and all trips, while on the right side we consider all trips and the arrival depot (t). Edges correspond to arcs in network G. The structure of the SDVSP corresponds to the assignment structure, except for depot nodes which may be assigned to several other nodes. The total number of assignments in $(s\times N)\cup(N\times t)$ of a feasible vehicle schedule equals the number of vehicles used in the schedule.

A more compact formulation than the quasi-assignment formulation, in terms of less variables, can be obtained by substitution. Consider network $G^T=(N,E)$ corresponding to a bipartite graph representation of the assignment problem, which is obtained from G by leaving out nodes s and t and the corresponding arcs. We can formulate the SDVSP using network G^T by substituting the variables corresponding to links with the depot out of the quasi-assignment model:

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

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

and define d_{ij} as the cost of linking trips i and j, which is obtained from c_{ij} as follows:

$$d_{ij} = c_{ij} - c_{sj} - c_{it} \qquad \forall\ (i,j) \in E.$$

That is, the costs for links with the depot is transferred to links between trips. The objective function becomes

$$\sum_{i,j\in A} c_{ij}y_{ij} = \sum_{i,j\in E} c_{ij}y_{ij} + \sum_{j\in N} c_{sj}y_{sj} + \sum_{i\in N} c_{it}y_{it} =$$

$$\sum_{i,j\in E} c_{ij}y_{ij} + \sum_{j\in N} c_{sj}\left(1 - \sum_{i:(i,j)\in E} y_{ij}\right) + \sum_{i\in N} c_{it}\left(1 - \sum_{j:(i,j)\in E} y_{ij}\right) =$$

$$\sum_{i,j\in E} d_{ij}y_{ij} + \sum_{j\in N} c_{sj} + \sum_{i\in N} c_{it}$$

and he SDVSP can be formulated as the two-sided inequality assignment model:

Two-Sided Inequality Assignment Model

minimize $\sum_{(i,j)\in E} d_{ij}\, y_{ij} + \sum_{j\in N} c_{sj} + \sum_{j\in N} c_{it}$

subject to

$$\sum_{j:(i,j)\in E} y_{ij} \le 1 \qquad \forall\ i \in N \qquad (2.2a)$$

$$\sum_{i:(i,j)\in E} y_{ij} \le 1 \qquad \forall\ j \in N \qquad (2.2b)$$

$$y_{ij} \in \{0,1\} \qquad \forall\ (i,j) \in E \qquad (2.2c)$$

The inequality constraints are obtained by observing that $y_{it} \ge 0$ for all $i \in N$ and $y_{sj} \ge 0$ for all $j \in N$. The intuitive idea of this model is that assignments become attractive when $d_{ij} \le 0$. Nodes that remain unassigned correspond to links with the depot. The constant terms can be omitted from the objective function and added to the solution afterwards. Bertsekas [3] does not propose a specialized algorithm for the *two-sided inequality constrained assignment problem*, but notes that this problem can be converted into the *asymmetric assignment problem* for which Bertsekas and Castañon [5] propose several algorithms. In the following we describe this conversion.

Consider network $G^S=(M,E^S)$ which is obtained from network G^T by introducing for each trip i an artificial node $n+i$ and a zero cost arc $(i, n+i)$. That is, we consider two node sets $N=\{1,...,n\}$ and $M=\{1,...,2n\}$ and the arc set $E^S=E\cup\{(i, n+i), i=1,...,n\}$. An assignment of a trip $i \in N$ to an artificial node $(n+i) \in M$ corresponds to a vehicle returning to the depot after covering trip i, while an unassigned trip $j \in M$ $(j \le n)$ corresponds to a vehicle leaving the depot to cover trip j. The costs b_{ij} are obtained from c_{ij} as follows:

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

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 an asymmetric assignment model:

Asymmetric Assignment Model

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

subject to

$$\sum_{j:(i,j)\in E^S} y_{ij} = 1 \qquad \forall\ i \in N \tag{2.3a}$$

$$\sum_{i:(i,j)\in E^S} y_{ij} \le 1 \qquad \forall\ j \in M \tag{2.3b}$$

$$y_{ij} \in \{0,1\} \qquad \forall\ (i,j) \in E^S \tag{2.3c}$$

Similar to the inequality assignment model, the constant terms can be omitted from the objective and added to the solution afterwards.

2. AN AUCTION ALGORITHM FOR THE QUASI-ASSIGNMENT PROBLEM

AUCTION ALGORITHMS ARE primal-dual algorithms for network flow problems mainly developed by Bertsekas [3]. They are iterative procedures, where both primal and dual costs may deteriorate at any one iteration, although in the end an optimal solution is obtained. Originally, auction algorithms were developed for parallel computation, but they turned out to be very fast in a sequential environment as well. We will discuss, respectively, the basic idea of auction algorithms for the assignment problem, and the complementary slackness conditions and the auction algorithm for the quasi-assignment problem.

2.1. Auction Algorithms for the Assignment Problem

To illustrate that auction algorithms are highly intuitive, we give a brief introduction of auction algorithms for the assignment problem. In the linear assignment problem (LAP) we want to find a complete one-to-one assignment of a number of persons to an equal number of objects, so as to minimize the total costs, where a cost is given for assigning a person to an object. Suppose that we are dealing with a maximization problem, replacing the cost of assigning a person to an object with a benefit defined by the negative of the cost. Auction algorithms exist in three forms: forward, reverse and combined forward and reverse

algorithms. Each object has a price and each person a profit. In forward auction, a person is assigned to an object with maximum value. This value is defined as the benefit of assigning an object to a person less the price of the object. The object's price is then raised as much as possible, while satisfying complementary slackness. Maximal total benefit is attained when each person is assigned to an object with maximum value on a one-to-one basis; The corresponding set of prices solves an associated dual problem. Alternatively, for reverse auctions persons and objects are exchanged: unassigned objects select a best person and raise his/her profit as much as possible. The combined forward and reverse auction algorithm is obtained by switching between forward and reverse auctions. Because several persons (objects) may compete for a smaller number of equally desirable objects (persons), cycling may occur. Therefore, a perturbation mechanism called ε-*complementary slackness* (ε-*CS*) is used. For good theoretical as well as practical performance, a technique called ε-*scaling* is used, which consist 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. Each scaling phase provides good initial prices for the next.

The practical performance of auction algorithms has shown to be very efficient. Computational experience in [3] shows that an auction algorithm for the LAP is considerably faster than, for example, the sequential shortest path algorithm. Bertsekas and Eckstein [2] show that the worst-case complexity of the forward auction algorithm for the LAP using ε-scaling is $O(nm\log nC)$, where m is the number of possible links between persons and objects and C is the maximum absolute benefit. The computational analysis performed by Schwartz [17] indicates an expected computational complexity of $O(n^2\log n)$ for this forward auction algorithm. No complexity results are known for the combined forward and reverse auction algorithm, but computational results in Bertsekas [3] have shown that the combined version is considerably faster than the forward version alone.

2.2. Complementary Slackness for the Quasi-Assignment Problem

We adapt a combined forward and reverse auction algorithm for the LAP as suggested by Bertsekas [3] for the quasi-assignment problem. The so-called *Gauss-Seidel* version is considered, where in each iteration unassigned trips *bid* sequentially for another trip or the depot. This version is efficient when only sequential implementations are considered. To keep the auction terminology we define the quasi-assignment problem (QAP) as the problem of assigning n persons to n objects, while persons can also be assigned to the *sink* (node t in network G) and objects to the *source* (node s in network G). A quasi-assignment is feasible if each person is either assigned to an object or to the sink, while each object is either assigned to a person or to the source. Therefore, we distinguish three types of assignments:
(i) person-object, denoted by set $S = \{(i,j) \mid (i,j) \in E \text{ and assigned}\}$;
(ii) source assignments, denoted by set $D_s = \{j \mid j \in N \text{ and assigned to the source}\}$;
(iii) sink assignments, denoted by set $D_t = \{i \mid i \in N \text{ and assigned to the sink}\}$.

A quasi-assignment is completely defined by $Q = S \cup D_s \cup D_t$. 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. Note that only the combined forward/reverse algorithm can be used for the QAP: in forward auction iterations persons can be assigned to objects or the sink, while in reverse auction iterations objects can be assigned to persons or the source. In the bipartite graph G in Figure 1 (Section 1), this corresponds to assignments from left to right in forward auctions,

and assignments form right to left in reverse auctions.

Recall that E is the set of arcs in A between trips only, which corresponds to person-object links. Furthermore, π_i (profit) and p_j (price) denote the dual variables corresponding to constraints (2.1a) and (2.1b), respectively. The dual of the linear programming formulation of the QAP is then formulated as follows:

minimize $\sum_{i \in N} \pi_i + \sum_{j \in N} p_j$

subject to

$$\pi_i + p_j \geq a_{ij} \qquad \forall\ (i,j) \in E \tag{3.3a}$$

$$\pi_i \geq a_{it} \qquad \forall\ i \in N \tag{3.3b}$$

$$p_j \geq a_{sj} \qquad \forall\ j \in N \tag{3.3c}$$

As for the LAP the concept of ε-*complementary slackness* (ε-*CS*) is used.

Definition 3.1
A quasi-assignment Q and a profit-price pair (π,p) satisfy ε-*CS* for the QAP if for $\varepsilon>0$:

$$\pi_i + p_j \geq a_{ij} - \varepsilon \qquad \forall\ (i,j) \in E; \tag{3.4a}$$

$$\pi_i + p_j = a_{ij} \qquad \forall\ (i,j) \in S; \tag{3.4b}$$

$$\pi_i \geq a_{it} \qquad \forall\ i \in N; \tag{3.4c}$$

$$\pi_i = a_{it} \qquad \forall\ i \in D_t; \tag{3.4d}$$

$$p_j \geq a_{sj} \qquad \forall\ j \in N; \tag{3.4e}$$

$$p_j = a_{sj} \qquad \forall\ j \in D_s; \tag{3.4f}$$

No perturbation mechanism is necessary for bids on the sink or the source. Because multi-assignments are allowed to the sink (source) two persons (objects) can not compete for it. For the LAP, Bertsekas [3] shows that a feasible assignment, which satisfies a similar ε-*CS* condition together with a price vector, is within $n\varepsilon$ of being optimal, where n is the number of persons or objects to be assigned. Then, to guarantee optimality of the final quasi-assignment solution it is sufficient that $\varepsilon<1/n$. For the QAP we can prove a similar condition.

Proposition 3.2
Suppose that a feasible quasi-assignment Q together with a profit-price pair (π,p) satisfy ε-*CS* for the QAP, then Q is within $n\varepsilon$ of being an optimal quasi-assignment. Moreover, define $\bar{p}_j = p_j+\varepsilon$ for all $j \in N$. Then the feasible dual solution given by $(\pi,\bar{p})$ is within $n\varepsilon$ of being optimal for the dual problem (3.3).

Proof
Let the given quasi-assignment Q be defined by $\{(i,j_i) \mid (i,j_i) \in S\} \cup \{j \mid j \in D_s\} \cup \{i \mid i \in D_t\}$ and

suppose that Q satisfies the ε-*CS* condition together with a profit-price vector pair (π,p). Then

$$\pi_i + \bar{p}_j = a_{ij} + \varepsilon \qquad \forall\ (i,j_i)\in S; \tag{3.7a}$$

$$\pi_i = a_{it} \qquad \forall\ i\in D_t; \tag{3.7b}$$

$$\bar{p}_j = a_{sj} + \varepsilon \qquad \forall\ j\in D_s. \tag{3.7c}$$

The primal value of Q is given by

$$P^Q = \sum_{i:\ (i,j_i)\in S} a_{ij_i} + \sum_{i\in D_t} a_{it} + \sum_{j\in D_s} a_{sj}$$

while the dual value corresponding to $(\pi,\bar{p})$ is given by

$$D^{(\pi,\bar{p})} = \sum_{i:\ (i,j_i)\in S} \pi_i + \sum_{i\in D_t} \pi_i + \sum_{i:\ (i,j_i)\in S} \bar{p}_{j_i} + \sum_{j\in D_s} \bar{p}_j$$

Adding (3.7a), (3.7b) and (3.7c) over all $(i,j_i)\in S$, $i\in D_t$ and $j\in D_s$ gives

$$\sum_{i:\ (i,j_i)\in S} \pi_i + \sum_{i\in D_t} \pi_i + \sum_{i:\ (i,j_i)\in S} \bar{p}_{j_i} + \sum_{j\in D_s} \bar{p}_j = \sum_{i:\ (i,j_i)\in S} a_{ij_i} + \sum_{i\in D_t} a_{it} + \sum_{j\in D_s} a_{sj} + n\varepsilon.$$

Let P^* and D^* be the optimal value of the primal and dual problem, respectively. Because of strong duality it holds that $D^*=P^*$. Then we obtain

$$D^* \le D^{(\pi,\bar{p})} = P^Q + n\varepsilon \le P^* + n\varepsilon,$$

where the first inequality holds because $(\pi,\bar{p})$ is feasible for the dual problem. Since $D^*=P^*$, it follows that the total benefit of the quasi-assignment is within $n\varepsilon$ of the optimal value P^*, while the dual cost of $(\pi,\bar{p})$ is within $n\varepsilon$ of the optimal dual cost D^*.

□

2.3. Forward/Reverse Auction Algorithm

Partial quasi-assignments are quasi-assignments where only a subset of the persons has been matched to objects or the sink. The combined forward/reverse auction algorithm for the QAP generates a sequence of profit-price vector pairs (π,p) and partial quasi-assignments, while keeping ε-*CS* satisfied in each iteration, until a feasible quasi-assignment is obtained.

Let $A(i)=\{j \mid (i,j)\in A\}$ denote the set of objects and the sink which are linked to person $i\in N$ in network G, and $B(j)=\{i \mid (i,j)\in A\}$ denote the set of objects and the source which are linked to object $j\in N$ in network G. In a forward auction iteration unassigned persons bid for objects or the sink. Initially, one uses an arbitrary set of prices together with the empty quasi-assignment. Let $f_{ij} = a_{ij}-p_j$ be the value of the bid of person i for object j and let $f_{it}=a_{it}+\varepsilon$ be the value of the bid of person i for the sink t, with $\varepsilon>0$ defined as before.

Typical Iteration of Forward Auction

Step 1. Find an unassigned person i, its best object or the sink

$$j_i = \operatorname*{argmax}_{j\in A(i)} f_{ij},$$

the corresponding values

$$v_i = \max_{j\in A(i)} f_{ij},$$

and, if $|A(i)|>1$,

$$w_i = \max_{j\in A(i),\ j\neq j(i)} f_{ij},$$

otherwise $w_i = -\infty$.
If $j_i \neq t$ then Step 2, otherwise Step 3.

Step 2. Update the prices:

$$p_{j_i} = p_{j_i} + v_i - w_i + \varepsilon = a_{ij_i} - w_i + \varepsilon, \tag{3.8}$$

$$\pi_i = a_{ij_i} - p_{j_i},$$

and update the assignments: (i,j_i) is added to S. If object j_i was already assigned to another person, this assignment is removed from S. Otherwise, if trip j_i was assigned to the source, this assignment is removed from D_s.

Step 3. Update the price: $\pi_i = a_{it}$, and update the assignment: add i to D_t.

□

Similarly, in the reverse auction iteration unassigned objects bid for persons or the source. Let $r_{ij} = a_{ij} - \pi_i$ be the value of the bid of object j for person i and let $r_{sj} = a_{sj}+\varepsilon$ be the value of the bid of object j for the source.

Typical Iteration of Reverse Auction

Step 1. Find an unassigned object j, its best person or the source

$$i_j = \operatorname{argmax}_{i\in B(j)} r_{ij},$$

the corresponding values

$$\beta_j = \max_{i\in B(j)} r_{ij},$$

and, if $|B(j)|>1$,

$$\omega_j = \max_{i\in B(j),\ i\neq i(j)} r_{ij},$$

otherwise $\omega_i = -\infty$.
If $i_j \neq s$ then Step 2, otherwise Step 3.

Step 2. Update the prices:

$$\pi_{i_j} = \pi_{i_j} + \beta_j - \omega_j + \varepsilon = a_{i_j j} - \omega_j + \varepsilon,$$

$$p_j = a_{i_j j} - \pi_{i_j},$$

and update the assignments: (i_j,j) is added to S. If person i_j was assigned to another object already, this assignment is removed from S. Otherwise, if person i_j was assigned to the sink, this assignment is removed from D_t.

Step 3. Update the price: $p_j = a_{sj}$, and update the assignment: add j to D_s.

□

To guarantee convergence of the algorithm, one refrains from switching between forward and reverse auctions until at least one more person or object becomes assigned. This is the case when, in a forward auction iteration a person i is assigned to a previously unassigned object or to the sink, or in a reverse auction iteration an object j is assigned to a previously unassigned person or to the source. We refer to a sequence of forward (backward) iterations until a switch occurs, as a forward (backward) pass. Note that with this rule at most n switches are possible. In the following proposition it is shown that this algorithm maintains ε-*CS* conditions (3.4) for the pair (π,p) throughout the algorithm.

Proposition 3.3
If the quasi-assignment and the profit-price pair available at the start of an iteration of either the forward or the reverse auction iteration satisfy the ε-*CS* conditions (3.4), the same is true for the quasi-assignment and the profit-price pair at the end of the iteration.

Proof
Consider a forward auction iteration, and let (π,p) and $(\bar{\pi},\bar{p})$ be the profit-price pair before and after the iteration, respectively. Then, $\bar{p}_j \geq p_j$ for all $j \in N$. Therefore, we have

$\bar{\pi}_i + \bar{p}_j \geq a_{ij} - \varepsilon$ for all $(i,j) \in E$ such that $\bar{\pi}_i = \pi_i$,
$\bar{\pi}_i \geq a_{it}$ for all $i \in N$ such that $\bar{\pi}_i = \pi_i$,
$\bar{p}_j \geq a_{sj}$ for all $j \in N$.

Furthermore, we have

$\bar{\pi}_i + \bar{p}_j = \pi_i + p_j = a_{ij}$ for all $(i,j) \in S$ assigned before as well as after the iteration;
$\bar{\pi}_i = a_{it}$ for all $i \in D_t$ assigned before as well as after the iteration.

What remains is to verify that the conditions

$$\bar{\pi}_i + \bar{p}_j \geq a_{ij} - \varepsilon \text{ for all } j \in A(i) \tag{3.9a}$$
$$\bar{\pi}_i \geq a_{it} \tag{3.9b}$$

hold for all persons i that submitted a bid and were assigned to j_i during the iteration.
If $j_i \neq t$ for such a person i we have, by inequality (3.8)

$$\bar{p}_{j_i} = a_{ij_i} - w_i + \varepsilon = a_{ij_i} - \max_{j \in A(i),\, j \neq j_i} f_{ij} + \varepsilon$$

which implies that

$$\bar{\pi}_i = a_{ij_i} - \bar{p}_{j_i} \geq f_{ij} - \varepsilon = a_{ij} - p_j - \varepsilon \geq a_{ij} - \bar{p}_j - \varepsilon \quad \text{for all } j \in A(i) \backslash \{j_i\}$$

and

$$\bar{\pi}_i = a_{ij_i} - \bar{p}_{j_i} \geq f_{it} - \varepsilon = a_{it}$$

and the conditions (3.9a) and (3.9b) hold.
If $j_i = t$ we have

$$\bar{\pi}_i = a_{it} \geq a_{ij} - p_j - \varepsilon \geq a_{ij} - \bar{p}_j - \varepsilon \quad \text{for all } j \in A(i)$$

and the conditions (3.9a) and (3.9b) hold again.
Similar arguments can be used to prove that ε-*CS* is maintained in a reverse auction iteration. □

We 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. The reason for this is that S is empty and bids only include p_j, while the update of p_j and π_i ensure that the ε-*CS* conditions are satisfied. Next, we address the finiteness of the algorithm.

Proposition 3.4
If at least one feasible quasi-assignment exists, the combined forward/reverse auction algorithm terminates after a finite number of iterations, and with a feasible assignment satisfying ε-*CS*.

Proof
If every forward and reverse auction pass terminates, then, with the switch rule described previously, the algorithm terminates. Consider a forward auction pass. Bertsekas [3] has shown that a finite number of bids occur from persons to objects in the LAP. We will not repeat the proof, but the same arguments can be used to show that there are only a finite number of bids of persons to objects in the QAP. Furthermore, it is easy to see that a finite number of bids occur from a person to the sink. Once a person is assigned to the sink the forward pass terminates, because multiple assignments to the sink are allowed. Similar arguments can be used to show that every reverse auction pass terminates. Therefore, the algorithm must terminate. The feasible assignment obtained upon termination satisfies ε-*CS* by Proposition 3.3.

□

3. DEALING WITH A LARGE NUMBER OF ARCS

A BOTTLENECK FOR vehicle scheduling applications when using the quasi-assignment network may be a large number of arcs. This occurs, for example, in urban bus scheduling where the average length of a trip is relatively short compared to the whole planning horizon. It is practical to assume that when a vehicle has a long idle time between two trips it will return to the depot. For example, the idle time may be long enough when the vehicle has time to return to the depot after the first trip, the crew has a break at the depot, and the vehicle can travel to the start location of the second trip. If we could simply replace the arcs corresponding to such a long idle time (*long arcs*) with the arcs for return to the depot, which already exist in the network, we could reduce the number of arcs in network G. However, in that case we can not put a fixed cost on arcs to or from the depot, because passing by the depot should not add the cost of using a vehicle. Reducing the number of long arcs may lead to a reduced computation time and/or reduced memory requirements. Although an efficient vehicle schedule generally contains a low percentage of long arcs, the schedule will often be of poor quality when not considering them at all.

One may tackle this problem heuristically by solving the model for several time periods in a first phase, and combining the schedules for the whole planning period in a second phase. However, there are several ways to deal efficiently with a large number of long arcs without loosing optimality. We will discuss two approaches to eliminate all long arcs by removing the fixed cost away from the arcs to and from the depot: a minimum cost flow approach and a quasi-assignment approach with an assumption about the cost structure. We end this Section with another way for dealing with a large number of long arcs, namely an arc generation approach. For the remainder of this Section, we assume that the primary

objective is to minimize the number of vehicles. This means that the fixed cost K added to the cost of arcs leaving and/or entering the depot is high enough to guarantee that this minimum number will be achieved. Define E^l as the set of long arcs in E, where $(i, j) \in E^l$ if and only if

$$et_i + trav(e_i, d) \leq bt_j - trav(d, b_j), \tag{4.1}$$

and E^s is E/E^l is the set of *short arcs*.

3.1. Minimum Cost Flow Approach

The use of a minimum cost flow formulation to remove long arcs is proposed by Bokinge and Hasselström [6] (see also Desrosiers et al. [10]). The minimum cost flow network $G^{mc}=(V^{mc},A^{mc})$ is obtained as follows: A trip $i \in N$ is represented by two nodes i' and i'' and an arc (i', i'') in between with zero cost and a minimal and maximal flow of one, thus assuring that it will be covered in any solution. Short arcs in E^s have a minimum flow of zero, a maximum flow of one and cost c_{ij} as before. To be able to replace long arcs by returns to the depot, the depot nodes s and t are now represented by a sequence of $2n$ depot nodes, one for each potential vehicle departure and arrival at the depot. In other words, corresponding to a trip $i \in N$ we have a *departure depot node* s_i and an *arrival depot node* t_i. The depot nodes are sorted by increasing time and linked by one arc from each depot node to the next one with a minimum flow of zero, a maximum flow of n and zero cost. A return arc from the last depot node to the first one with a minimum flow of zero, a maximum flow of n and the fixed cost K, ensures the conservation of flow and counts the number of vehicles required to operate the schedule which is guaranteed to be minimum. Let DP^{mc} denote the set of these arcs between depot nodes. The depot nodes are linked to trips by one arc from the departure depot node to the start node of the trip with a minimum flow of zero, a maximum flow of one and cost c_{si}, and by one arc from the end node of the trip to the arrival depot node with a minimum flow of zero, a maximum flow of one and cost c_{it} without the fixed cost K. Concluding, network G^{mc} is defined by

$$V^{mc}=\{(i', i'', s_i, t_i) \text{ for all } i \in N\} \text{ and}$$
$$A^{mc}=E^s \cup \{(i', i''), (i'', t_i), (s_i, i') \text{ for all } i \in N\} \cup DP^{mc}.$$

The size of the network can be reduced by grouping depot nodes together as follows: Depot nodes are examined in chronological order and a new group is started each time a arrival depot node is encountered, provided that in the current group there already exists a departure depot node. Each group of nodes is replaced by one new node, and the arcs between nodes in the group and nodes not in the group are replaced by arcs to and from the new node. This aggregation rule ensures that a vehicle leaving a depot node has already arrived at a previous depot node. Note that compared to network G, this network has generally less arcs because long arcs are not included, but more nodes because trip nodes are duplicated and several depot nodes are added.

3.2. Special Cost Structure

We propose a procedure to remove long arcs, which is only valid under certain cost assumptions. We will show that if the cost of deadheading trips is a linear function of travel plus waiting time, then there is no conflict between the two objectives of minimizing fleet

size and minimizing the cost of operating a schedule (see also Scott [18] for similar results). Define $G^s=(V,A^s)$ with $A^s=E^s\cup(s\times N)\cup(N\times t)$ and assume that the costs are as follows (possibly after scaling):

$c_{ij}=bt_j-et_i$ for each $(i,j)\in E^s$,
$c_{sj}=trav(d,b_j)$ for each $i\in N$,
$c_{it}=trav(e_i,d)$ for each $i\in N$,
$c_{ij}=c_{it}+c_{sj}$ for each $(i,j)\in E^l$.

The procedure for solving the SDVSP under these cost assumptions consists of building blocks in a first phase and combining those blocks in a second phase.

3.2.1. Phase I - Building Blocks

Sets of blocks are determined in network G^s in a similar way as vehicle schedules are determined in network G. A path from s to t in network G^s corresponds to one block, while such a path in network G corresponds to a schedule for a vehicle which may contain one or more blocks. Because there is no cost for waiting at the depot, all variable costs are determined by the blocks, and, when minimizing variable costs in G^s, we are minimizing variable costs in G. As before, we want to find a set of paths in G^s such that each trip is covered exactly once, which can be done by applying a quasi-assignment or asymmetric assignment algorithm.

Next, we show that the blocks produced in this way, can be combined to obtain a vehicle schedule with a minimum number of vehicles. Consider network G^v which is obtained from network G by leaving out the fixed cost on the arcs entering the sink. Our analysis relies on the concept of augmenting paths and cycles in a residual network (see, e.g., Ahuja et al. [1]). The residual network $G^v(Q)$ corresponding to a vehicle schedule Q is obtained from network G^v 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 forward arc corresponds to adding a new link between trips, while a backward arc corresponds to deleting a previous link. A path in network $G^v(Q)$ is said to be *alternating* with respect to schedule Q, if its arcs are alternately forward and backward. A forward arc into a node must be followed by a backward arc to un-assign its original predecessor, while a backward arc into a node must be followed by a forward arc to reassign its original successor. Each alternating path must start in s and end in t, and corresponds to a feasible change with respect to schedule Q by adding one extra vehicle. Such a path is called *augmenting*. An alternating cycle is an alternating path that starts and ends at the same node. A cycle is not augmenting, since it does not correspond to an extra vehicle. We are now able to prove the following important result.

Proposition 4.1
The blocks obtained in the way described above can be combined to obtain the minimum number of vehicles.

Proof
Suppose Q^{opt} is optimal in the sense that the number of vehicles is minimized, and that, given this number of vehicles, the variable cost is minimized. If the proposition does not hold, then there exists a vehicle schedule Q^{aug} with lower or equal variable cost and a larger number of vehicles. Of course, we assume that this number of vehicles is not larger than necessary. The

existence of Q^{aug} implies that there is at least one augmenting path with non-positive cost in $G^v(Q^{opt})$. Let $s \rightarrow i_1 \leftarrow i_2 \rightarrow ... \rightarrow i_{p-1} \leftarrow i_p \rightarrow t$ be such a path. It holds that $(i_p,i_1) \notin E^l$, because otherwise the block ending with $i_p \rightarrow t$ could be combined with the block starting with $s \rightarrow i_1$. This contradicts our assumption that the number of vehicles used in Q^{aug} is not larger than necessary. The proof that such an augmenting path AP with non-positive cost does *not* exist, is built up in three steps:

(i) Consider an augmenting path AP_1 with non-positive cost, and without forward and backward long arcs. 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

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

$$trav(d,b_{i_1}) - (bt_{i_1} - et_{i_2}) + bt_{i_3} - et_{i_2} - ... + bt_{i_{p-1}} - et_{i_{p-2}} - (bt_{i_{p-1}} - et_{i_p}) + trav(e_{i_p}) =$$

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

For AP_1 to have non-positive cost, we must have

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

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

(ii) Consider an augmenting path AP_2 with non-positive cost, and without forward long arcs. Let R^- be the set backward long arcs in AP_2 and define $z_{ij} = bt_j - et_i - trav(e_i,d) - trav(d,b_j)$ for each $(j,i) \in R^-$. Because (i,j) is a long arc, we know that $z_{ij} \geq 0$. Similar computations as in (i) result in:

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

For $c(AP_2)$ to be non-positive, $bt_{i_1} - et_{i_p} - trav(e_{i_p},d) - trav(d,b_{i_1})$ must be non-positive because $z_{ij} \geq 0$ for all $(i,j) \in E^l$. But then $(i_p,i_1) \in E^l$, which again is a contradiction.

(iii) Consider a general augmenting path AP, which may contain any type of arc. Because the cost of each forward long arc in AP is the sum of the corresponding travel times to t and from s, we can split AP at each forward long arc. Then, we obtain a set of paths of type (i) or (ii), of which each has positive cost. Thus, AP has positive cost.

Concluding, an augmenting path AP with non-positive cost does not exist.

□

3.2.2. Phase II - Combining Blocks

A quasi-assignment or asymmetric assignment algorithm is used in Phase I to determine a set of blocks B with cost $v(B)$. In Phase II, these blocks are combined, to obtain a complete vehicle schedule Q with cost $v(Q)$ and a minimum number of vehicles. Note that the number of vehicles is minimized when the number of links between blocks is maximized,

because every extra link means one vehicle less. Define the bipartite graph $G^b=(N_1 \cup N_2 , L)$, where $N_1=N_2=B$ and $L=\{(i,j): i,j\in B,\ i\prec j,\ i$ and j can be combined$\}$. It is easy to see that a feasible combination of blocks corresponds to a matching in graph G^b. Therefore, the problem of combining blocks such that the minimum number of vehicles will be attained, is a bipartite cardinality matching problem (see, e.g., Gerards [12]). This problem can be solved in $O(|L|\sqrt{|B|})$ time using a maximum flow algorithm. However, because of the special structure of this problem, we can use a simple greedy algorithm. The blocks in B are sorted by increasing starting time; ties are broken arbitrarily. Let $l(i)$ be the first block in B which can be served by the same vehicle after serving i; $l(i)$ equals zero if no block can be served after block i. We propose the greedy algorithm COMBINE below:

Algorithm COMBINE

Step 0. $Q \leftarrow \emptyset$;
$v(Q) \leftarrow v(B)$;
Sort the blocks in B by increasing starting time;

Step 1. While $B\neq\emptyset$ do
begin
  i is the first block in B;
  $B \leftarrow B/\{i\}$;
  $j \leftarrow l(i)$;
  while $j\neq 0$ do
  begin
    $i \leftarrow$ Merge(i, j);
    $B \leftarrow B/\{j\}$;
    $j \leftarrow l(j)$;
  end;
  $Q \leftarrow Q\cup\{i\}$
  $v(Q) \leftarrow v(Q)+K$;
end;

□

Merge(i,j) returns the block obtained by merging i and j together. The fixed cost K is added for each vehicle in use. This algorithm has a time complexity of $O(|B|^2)$, which can be reduced further by using advanced data structures. However, it is not worthwhile to consider such data structures because the running time of the algorithm appears to be very low in our computational tests. Next, we proof that this greedy procedure will always minimize the number of vehicles.

Proposition 4.2
The COMBINE algorithm will minimize the number of vehicles.

Proof
The proof is established by using matching theory. A path P in graph G^b is said to be alternating with respect to a matching M, or *M-alternating*, if the edges of P are alternatively in M and not in M. Nodes not matched by M are called *exposed*, and an alternating path P is *M-augmenting* if both ends are exposed. It is known from matching theory (see, e.g., [12])

that a matching M in a graph G^b is a maximum matching if and only if there is no M-augmenting path in G^b. Because a maximum matching in G^b corresponds to a minimum number of vehicles, we need to show that no M^C-augmenting path exists for the matching M^C obtained by COMBINE. If such a path with k edges exists it has the form $i_1 - j_1 - i_2 - j_2 - \ldots - i_k - j_k$, where $i_p \in N_1$ and $j_p \in N_2$ for $p=1,\ldots,k$. Note that because blocks in P are sorted by increasing starting time, we know that if $(i,j) \in L$, then $(i,k) \in L$ for $k=j+1,\ldots,|B|$. Because j_k is exposed we know that $(i_k, j_{k-1}) \in M^C$, and $j_{k-1} \prec j_k$ because otherwise the algorithm would have chosen $(i_k, j_k) \in M$. In the same way we can show that $j_p \prec j_k$ for $p=1,\ldots,k-2$. But if $j_1 \prec j_k$ and $(i_1,j_1) \in L$ then $(i_1, j_k) \in L$, which is not possible because i_1 and j_k are exposed and the algorithm would have chosen (i_1, j_k) in the solution. Therefore, we can conclude that there is no M-augmenting path in G^b and the minimum number of vehicles is obtained with COMBINE.

□

3.3. Arc Generation

Next, we discuss an arc generation approach for dealing with long arcs. This approach is slightly more general because, as opposed to the former two approaches, we may for example include idle time in the cost of a long arc. This arc generation approach is similar to a column generation approach for linear programming. Consider the quasi-assignment approach. The set A of arcs in network G can be divided into two sets, say A^b and A^e, for example corresponding to A^s and E^l defined previously. This suggests the following arc generation approach, where we consider the QAP again as a maximization problem:

Algorithm Arc Generation

Step 1. Apply the combined forward/reverse auction algorithm of Section 2.3 to the reduced network $G^b=(V,A^b)$. The resulting dual profit-price vector pair is (π,p).

Step 2. Calculate the reduced costs for each arc $(i,j) \in A^e$: $\bar{a}_{ij} = a_{ij} - \pi_i - p_j - \varepsilon$. For each trip $i \in N$, add to A^b one or more arcs $(i,j) \in A^e$ with positive reduced cost. If arcs are generated, delete the arcs just added from A^e and return to step 1. Otherwise, stop.

□

After the first time Step 1 is called, reoptimization is used by deleting all assignments and initializing the dual variables with the optimal values of the previous call to the algorithm. When no arcs are generated in step 2, the algorithm stops with the optimal solution, since the ε-complementary slackness conditions are satisfied for the complete network G. The key to this algorithm is that one hopes that few arcs need to be generated to find the optimal solution and that total computing time remains low due to fast reoptimization. This arc generation method can be varied by redefining the set A^e by adding, for example, more deadheading trips besides those corresponding to long arcs, and starting with a smaller set A^b. The arc generation algorithm also suggests a heuristic approach for very large instances that need a suboptimal solution fast, or where the complete network does not fit in CPU memory. The idea is that one stops after a few iterations of the arc generation algorithm.

4. COMPUTATIONAL RESULTS

WE COMPARE PERFORMANCE of several algorithms for the SDVSP using both randomly generated data and real life data. This data covers a wide range of applications of the SDVSP, although one has to take care when drawing conclusions on randomly generated data. We implemented the following algorithms in the C programming language:

- The Hungarian algorithm for the quasi-assignment problem (HUNG) proposed in Paixão and Branco [14]. The code is a C translation of the FORTRAN implementation by Branco [7].
- The forward/reverse auction algorithm for the quasi-assignment problem (QAFR) proposed in Section 2.3.
- A forward/reverse auction algorithm for the asymmetric assignment problem (ASFR), similar to algorithm asfr2 proposed in Bertsekas and Castañon [5].
- The minimum cost flow algorithm (MICO) described in Section 3.1 of this paper. The minimum cost flow code is a C translation (line-by-line) of the public domain FORTRAN implementation RELAX4 of a relaxation algorithm by Bertsekas and Tseng (see [3]).
- The arc generation algorithm (AGFR) proposed in Section 3.3. The set A^b is chosen as a subset of the set of short arcs: For each trip, adjacent trips are considered in order and all short arcs up to the first long arc are included in A^b.
- HUNG2, QAFR2 and ASFR2: the algorithm proposed in Section 3.2 with HUNG, QAFR and ASFR, respectively, to obtain blocks and algorithm COMBINE to combine the blocks.

All algorithms were run on a 486/66Mhz personal computer. For the algorithms that assume integer costs we can deal with rational costs by scaling. Here we assume all data to be integer. 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 down to $1/n$. 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 $(i,j)\in A$ ($\frac{1}{2}n$ for ASFR). Then, optimality is assured when the algorithm converges with ε=1. Integers may get out of range, but Bertsekas [4] 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 [3] for the LAP, is determined according to the following rule in each scaling phase i:

$$\varepsilon(i) = \max\{1,\Lambda/\theta^i\},$$

where $\Lambda>0$ and $\theta>1$. For QAFR, QAFR2, ASFR and ASFR2 we used the values $\Lambda = nC$ and $\theta = 4$, while for AGFR we used $\Lambda = nC/10$ and $\theta = 4$. These values are obtained as a result of intensive testing, and gave the globally fastest running times.

We consider randomly generated data and real life data. The random data is generated as suggested in Dell'Amico et al. [9], 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. There are v relief points, the coordinates of which are integers generated from a uniform distribution in a $60x60$ square, while v is an integer generated from a uniform distribution in

$\{\lceil n/3\rceil,\lceil n/3\rceil+1,...,\lceil n/2\rceil\}$. All times are given in minutes. For each pair (a, b) of relief points, the travel time from a to b, denoted by $trav(a, b)$, is given by the Euclidean distance between these points rounded to the nearest integer. For each trip $j \in N$, the starting and ending relief points, b_j and e_j, are generated as uniformly distributed in $\{1,2,... ,v\}$. The location of the depot is also a uniformly distributed in $\{1,2,...,v\}$. The starting and ending times of trip j are generated taking into account two classes of trips:

1. 40% of the trips are short trips: The starting time bt_j of trip j is generated as a uniformly distributed in {420,421,...,479} with a probability of 15%, in {480,481,...,1019} with a probability of 70%, and in {1020,1021,...,1080} with a probability of 15%. The ending time et_j of trip j is uniformly distributed in $\{bt_j+trav(b_j, e_j)+5, bt_j+trav(b_j, e_j)+6, ..., bt_j+trav(b_j, e_j)+40\}$.
2. 60% of the trips are long trips: The starting and ending time bt_j and et_j are uniformly distributed in {300,301,...,1200} and $\{bt_j+180, bt_j+181, ..., bt_j+300\}$, respectively. Moreover, every long trip j is a round trip, i.e. $b_j=e_j$.

The costs in terms of the quasi-assignment network are defined as:

1. c_{ij} = minimum[10 $trav(e_i, b_j)$ + 2(bt_j - et_i - $trav(e_i, b_j)$), 10($trav(d, b_j)+trav(e_i, d)$)] for all compatible pairs (i,j);
2. c_{si} = 10 $trav(d, b_i)$ for all trips i;
3. c_{jt} = 10 $trav(e_j, d)$ + 2,000 for all trips j.

For the costs between trips, the minimum is attained for the first term in case of short arcs, and for the second term in case of long arcs. The cost structure is such that the model finds the best solution (in terms of vehicle operating costs) with the minimum number of vehicles which ensures feasibility. Note that the special cost structure defined in Section 3.2 does not apply here. For these test problems we compare the algorithms HUNG, QAFR, ASFR, MICO and AGFR. For each value of n, varying from 25 to 1,000, ten instances of the SDVSP were generated. The first three columns in Table I show the value of n and the averages over these ten problems for the cost of the optimal solution and the corresponding number of vehicles. The remaining columns show the average and maximal CPU times in seconds, excluding input and output, for the five algorithms considered. The density of network G is defined by $m/(n+1)^2 * 100\%$, and is around 30% for these problems.

TABLE I
Randomly Generated Test Problems

n	cost	veh	Time (seconds)									
			HUNG		QAFR		ASFR		MICO		AGFR	
			mean	max	mean	max	mean	max	mean	max	mean	max
25	25,573	9	0.01	0.01	0.01	0.05	0.01	0.01	0.02	0.06	0.03	0.27
50	52,238	16	0.01	0.05	0.01	0.06	0.02	0.06	0.03	0.06	0.08	0.22
75	69,760	23	0.07	0.11	0.04	0.06	0.11	0.11	0.05	0.17	0.34	0.66
100	86,934	29	0.18	0.22	0.08	0.11	0.18	0.66	0.31	0.60	0.40	1.26
150	126,055	41	0.59	0.66	0.29	0.44	0.36	0.50	0.69	1.05	1.63	4.06
200	168,274	56	1.33	1.48	0.64	1.16	0.80	1.10	1.45	2.15	5.53	15.87
250	191,402	65	2.57	2.75	1.23	1.60	1.57	2.03	2.29	4.39	9.10	15.55
300	231,534	80	4.12	4.72	2.33	4.84	2.45	4.12	3.01	4.94	8.95	14.94
400	310,951	106	9.40	9.99	5.25	7.42	6.49	11.37	7.98	15.49	27.91	71.01
500	359,737	125	16.92	19.06	15.62	46.91	15.07	33.18	10.93	16.91	59.06	224.99
750	537,032	186	53.46	59.21	42.69	60.09	47.35	62.94	38.15	85.85	119.96	236.84
1,000	716,182	244	118.85	134.01	115.16	158.07	178.33	269.69	81.14	147.09	394.97	691.18

In Table I we can see that up to n is 400, QAFR is the fastest algorithm, while MICO performs better for the larger sized instances. Note that HUNG is very stable with maxima close to the averages. An explanation might be that this is the only algorithm that does not use scaling. To see if we could improve the running times of AGFR, we tried to use the Hungarian algorithm in the arc generation instead of the auction algorithm, but the performance was much worse. Corresponding to the tests reported in Table I, we show in Table II the number of trips and the average number of arcs in G, the minimum cost flow network G^{mc}, and the number of initial arcs and arcs generated for the arc generation procedure.

TABLE II
Number of Nodes and Arcs

network G		network G^{mc}		arc generation	
n	m	$n(G^{mc})$	$m(G^{mc})$	initial	generated
25	196	60	157	60	46
50	773	117	463	170	130
75	1,697	176	922	335	204
100	3,051	234	1,590	590	220
150	6,717	347	3,611	1,421	264
200	12,546	455	5,505	2,142	712
250	19,618	562	8,261	3,061	980
300	27,895	671	10,936	4,699	1,111
400	49,407	886	23,372	10,086	1,143
500	77,131	1,100	28,873	7,551	2,773
750	173,069	1,619	74,813	29,024	3,565
1,000	305,963	2,125	141,162	44,608	5,416

Note from Table II the increase in the number of nodes and the decrease in the number of arcs for the minimum cost flow network compared to network G. Also note the relatively few arcs needed to detect optimality for the arc generation algorithm AGFR. We tried the following three variations of the random data generation:

- simulating urban mass transit by generating only short trips which increases the network density to about 39%,
- simulating suburban/intercity mass transit by generating only long trips which decreases the network density to about 25%, and
- increasing fixed cost to 5,000.

The last variation is used to test for the cost sensitivity of the algorithms. Table III shows the average computation times in seconds over ten instances for n=500, 750 and 1,000. For the urban variation with n=1,000 it was not possible to obtain results with MICO due to memory limitations.

TABLE III
Variations Random Data

		Time (seconds)				
variation	n	HUNG	QAFR	ASFR	MICO	AGFR
urban	500	27	17	17	22	155
	750	84	63	54	45	801
	1,000	182	154	200	--	--
suburban	500	10	14	18	11	128
	750	25	39	90	37	128
	1,000	55	82	282	73	256
K=5,000	500	17	14	18	12	69
	750	52	47	50	34	145
	1,000	116	150	224	81	380

Note that the relative performance improves for QAFR for the urban situation and for HUNG for the suburban situation. We redefine the costs in order to test problems with the special cost structure defined in Section 3.2. Recall that the algorithm proposed in Section 3.2 only works when cost is a linear function of travel plus waiting time. The costs are now given by:

1. $c_{ij} = bt_j - et_i$ for all $(i,j) \in E^s$;
2. $c_{si} = trav(d, b_i) + 1{,}000$ for all trips i;
3. $c_{jt} = trav(e_j, d) + 1{,}000$ for all trips j,

We compare the algorithms HUNG2, QAFR2, ASFR2 and MICO. Although there are no long arcs MICO is still included for the sake of comparison. For each value of n, varying from 25 to 2,000, ten instances of the SDVSP were generated. The first three columns in Table IV show the value of n and the averages over these ten problems for the cost of the optimal solution and the corresponding number of vehicles. The remaining columns show the mean and maximal CPU times in seconds, excluding input and output, of the four algorithms considered. The density of network G for these problems is around 3%. Compare this with the 35% density of network G for random problems with long arcs. For the two largest instances it was not possible to obtain results with MICO due to memory limitations.

TABLE IV
Randomly Generated Test Problems - Special Cost Structure

			Time (seconds)							
n	cost	veh	HUNG2		QAFR2		ASFR2		MICO	
			mean	max	mean	max	mean	max	mean	max
25	18,503	8	0.01	0.05	0.01	0.06	0.01	0.06	0.01	0.06
50	37,960	17	0.01	0.06	0.01	0.05	0.01	0.05	0.01	0.06
75	48,059	22	0.01	0.06	0.01	0.05	0.01	0.05	0.03	0.06
100	64,783	30	0.04	0.10	0.01	0.05	0.01	0.10	0.07	0.16
150	96,307	45	0.07	0.11	0.02	0.12	0.03	0.11	0.19	0.28
200	125,163	58	0.13	0.22	0.01	0.06	0.04	0.12	0.27	0.55
250	146,606	68	0.32	0.44	0.06	0.11	0.30	1.37	0.50	0.77
300	174,222	81	0.50	0.88	0.11	0.27	0.39	1.43	0.81	1.32
400	212,200	98	1.14	1.64	0.26	0.44	0.63	1.21	1.85	2.58
500	273,779	128	1.45	2.36	0.32	0.60	2.24	4.72	2.42	3.30
750	403,430	188	4.64	6.76	1.34	2.37	13.63	24.00	7.97	14.22
1,000	526,960	246	9.97	13.12	3.28	5.11	35.97	50.03	16.64	21.75
1,500	823,609	386	28.19	43.17	10.94	23.35	137.08	263.42	--	--
2,000	1,091,730	511	54.50	94.26	26.29	68.60	307.20	737.98	--	--

We can see that algorithm QAFR2 performs considerably better than the other algorithms, both with respect to average and maximum running times. ASFR2 performs poorly on the cases with n=750 and n=1,000, which can be verified by the fact that MICO performs considerably better.

In addition to these randomly generated problems, we used two sets of data from real life cases. The first set are instances from the CARRIS public transport company of Lisbon, Portugal, which correspond to clusters of bus lines covering peak hours only. We used the following cost definitions for the CARRIS data:

1. $c_{ij} = trav(e_i, b_j) + (bt_j - et_i)$ for all compatible pairs (i,j);
2. $c_{si} = trav(d, b_i) + 1{,}000$ for all trips i;
3. $c_{jt} = trav(e_j, d) + 1{,}000$ for all trips j.

Because, due to the short planning horizon, there are no long arcs in the assignment networks, only the assignment based algorithms HUNG, QAFR and ASFR are of interest. The results are shown in Table V. For each value of the number of trips n, the Table shows the number of arcs m in G, the cost of the optimal solution, and the corresponding number of vehicles. The last three columns show the CPU times in seconds, excluding input and output, for the three algorithms considered. The density of G for these problems is below 8%.

TABLE V
Real Life Test Problems - CARRIS

				Time (seconds)		
n	*m*	cost	vehicles	HUNG	QAFR	ASFR
192	1,640	101,482	49	0.16	0.05	0.06
192	2,961	101,435	49	0.22	0.05	0.11
255	2,920	134,370	65	0.28	0.22	0.22
255	5,200	132,250	64	0.44	0.17	0.21
264	1,934	102,703	50	0.17	0.01	0.05
264	2,761	100,639	49	0.22	0.01	0.11
352	857	139,653	68	0.22	0.01	0.05
352	4,026	133,487	65	0.44	0.05	0.11

Algorithm QAFR outperforms the other two algorithms for these instances, although all three algorithms run fast due to the low density of the networks. Additionally, we tested HUNG2, QAFR2, ASFR2 and MICO for instances from the RET public transport company of Rotterdam, the Netherlands, which correspond to bus lines or clusters of bus lines covering one day. The cost definitions are similar to those of the randomly generated tests with the special cost structure. The results are presented in Table VI. Note the low density (in %) of network *G* in the second column. The relative performance of the algorithms is similar to the performance Table IV. For these test problems, QAFR2 and ASFR2 are extremely fast. The reason for this may be 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.

TABLE VI
Real Life Test Problems - RET

				Time (seconds)			
n	dens.	cost	vehicles	HUNG2	QAFR2	ASFR2	MICO
72	2.68	11,351	5	0.01	0.01	0.01	0.01
113	2.74	17,156	8	0.05	0.01	0.01	0.11
126	1.68	8,627	4	0.06	0.01	0.05	0.05
127	1.40	10,879	5	0.01	0.01	0.01	0.06
148	3.05	23,422	11	0.01	0.01	0.01	0.11
171	1.62	12,815	6	0.06	0.06	0.01	0.11
197	1.19	16,962	8	0.06	0.05	0.01	0.16
238	1.14	23,139	11	0.11	0.01	0.01	0.22
248	1.67	21,093	10	0.11	0.01	0.06	0.33
386	1.12	46,299	22	0.44	0.01	0.05	0.66
486	1.11	44,230	21	0.66	0.05	0.06	0.88
634	0.99	67,391	32	1.27	0.06	0.11	1.92
949	0.30	111,310	53	1.93	0.01	0.11	1.70
1096	0.26	134,440	64	2.59	0.05	0.06	1.98
1328	0.22	157,491	75	3.57	0.05	0.11	2.53

5. CONCLUSIONS

WE HAVE PROVIDED the reader with an overview of formulations and algorithms for the SDVSP, including new models and algorithms. New algorithms are based on recent successful developments with respect to auction algorithms for network flow problems. The computational study provides a tool for OR scientists and practioners for deciding which algorithm to use for their purposes. This may depend on the cost structure, and weather the SDVSP is used as a standalone problem, or as a subproblem in more complex scheduling problems. We have performed tests for different settings, such as urban bus scheduling and suburban bus scheduling, and using both randomly generated and real life data.

Since for many applications it is realistic to choose the special cost structure defined in Section 3.2, the fastest algorithm to use is QAFR. However, for cases where the special cost structure can not be used, for example when the SDVSP appears as a subproblem, a choice must be made between QAFR or the minimum cost flow approach (MICO). The tests indicate that for smaller sized instances it is better to use QAFR, while for larger sized instances it is better to use MICO. Computational speed might not be the only criterium. For

example, when memory requirements play a role QAFR is preferred. Concluding, when considering computation speed and memory requirements, the forward/reverse auction algorithm QAFR is the best algorithm overall. Its performance can be improved further by using problem specific ε-scaling, because we have used the same parameters for all test cases in our computational study. Although the performance of the arc generation approach is rather disappointing, it remains an interesting topic for future research. Performance may for example be improved by adapting scaling parameters during the various reoptimizations.

ACKNOWLEDGEMENTS

The authors are grateful to the RET company in Rotterdam and the CARRIS company in Lisbon for providing them with the data.

REFERENCES

1. R.K. Ahuja, T.L. Magnanti and J.B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, New Jersey, 1993.
2. D.P. Bertsekas and J. Eckstein, "Dual Coordinate Step Methods for Linear Network Flow Problems," *Mathematical Programming* 42, 203-243 (1988).
3. D.P. Bertsekas, Linear Network Optimization: Algorithms and Codes, MIT Press, Cambridge, MA, 1991.
4. D.P. Bertsekas, "Auction Algorithms for Network Flow Problems: A Tutorial Introduction," *Computational Optimization and Applications* 1, 7-66 (1992).
5. D.P. Bertsekas and D.A. Castañon, "A Forward/Reverse Auction Algorithm for Asymmetric Assignment Problems," *Computational Optimization and Applications* 1, 277-297 (1992).
6. U. Bokinge and D. Hasselström, "Improved Vehicle Scheduling in Public Transport through Systematic Changes in the Time-Table," *European Journal of Operations Research* 5, 388-395 (1980).
7. I.M. Branco, "Algoritmos para Modelos Matemáticos de Quasi-Afectação e Extensões," Phd Thesis (in Portuguese), DEIO - FCUL, Universidade de Lisboa, Portugal, 1989.
8. J.R. Daduna and J.M. Pinto Paixão, "Vehicle Scheduling for Public Mass Transit - An Overview," in *Computer-Aided Transit Scheduling: Proceedings of the Sixth International Workshop*, J.R. Daduna, I.M. Branco and J.M. Pinto Paixão (eds), Springer Verlag, Berlin, 1995, pp 76-90.
9. M. Dell'Amico, M. Fischetti and P. Toth, "Heuristic Algorithms for the Multiple Depot Vehicle Scheduling Problem", *Management Science* 39, 115-125 (1993).
10. J. Desrosiers, Y. Dumas, M.M. Solomon and F. Soumis, "Time Constrained Routing and Scheduling," in *Handbooks of Operations Research and Management Science, Volume 8: Network Routing*, M.O. Ball et al. (eds.), North-Holland, Amsterdam, 1995, pp. 35-139.
11. R. Freling, C.G.E. Boender and J.M. Pinto Paixão, "An Integrated Approach to Vehicle and Crew Scheduling," Report 9503/A, Econometric Institute, Erasmus University Rotterdam, 1995.
12. A.M.H. Gerards, "Matching," in *Handbooks in Operations Research and Management*

Science, Volume 7: Network Models, M.O. Ball et al. (eds.), North Holland, Amsterdam, 1995, pp. 135-224.
13. R. Jonker and T. Volgenant, "Improving the Hungarian Assignment Algorithm", *Operations Research Letters* 5, 171-176 (1986).
14. J.M. Pinto Paixão and I.M. Branco, "A Quasi-Assignment Algorithm for Bus Scheduling," *Networks* 17, 249-269 (1987).
15. J.M. Pinto Paixão and I.M. Branco, "Bus Scheduling with a Fixed Number of Vehicles," in *Computer-Aided Transit Scheduling: Proceedings of the Fourth International Workshop*, J.R. Daduna and A. Wren (eds), Springer Verlag, Berlin, 1988, pp 28-40.
16. C.C. Ribeiro and F. Soumis, "A Column Generation Approach to the Multiple-Depot Vehicle Scheduling Problem," *Operations Research* 42, 41-52 (1994).
17. B.L. Schwartz, "A Computational Analysis of the Auction Algorithm," *European Journal of Operations Research* 74, 161-169 (1994).
18. D. Scott, "Minimal Fleet Size in Transhipment-Type Vehicle Scheduling Problems," Publication #487, Centre de Recherche sur les Transports, Univ. de Montréal, Que., 1986.
19. T. Song and L. Zhou, "A New Algorithm for the Quasi-Assignment Problem," *Annals of Operations Research* 24, 205-223 (1990).