DRIVE: DYNAMIC ROUTING OF INDEPENDENT VEHICLES

MARTIN SAVELSBERGH

Georgia Institute of Technology, Atlanta, Georgia

MARC SOL

Baan Company, Ede, The Netherlands

(Received April 1996; revision received July 1996; accepted January 1997)

We present DRIVE (Dynamic Routing of Independent VEhicles), a planning module to be incorporated in a decision support system for the direct transportation at Van Gend and Loos BV. Van Gend and Loos BV is the largest company providing road transportation in the Benelux, with about 1400 vehicles transporting 160,000 packages from thousands of senders to tens of thousands of addressees per day. The heart of DRIVE is a branch-and-price algorithm. Approximation and incomplete optimization techniques as well as a sophisticated column management scheme have been employed to create the right balance between solution speed and solution quality. DRIVE has been tested by simulating a dynamic planning environment with real-life data and has produced very encouraging results.

Van Gend and Loos BV (a member of the Nedlloyd group, a Dutch organization providing world-wide transportation services by land, air, and sea) is the largest company providing road transportation in the Benelux, which is the region consisting of Belgium, The Netherlands, and Luxembourg, with about 1400 vehicles transporting 160,000 packages from thousands of senders to tens of thousands of addressees per day. The service of Van Gend and Loos BV can be roughly divided into two parts, the *regular transportation system* and the *direct transportation system*. In the regular transportation system, shipments ranging from small packages to loads of up to four pallets are picked up at the sender and then delivered at the closest distribution center. During the night, the loads are transported from that distribution center to the distribution center closest to the destination of the load, from where they are delivered at their final destination during the next day. In the direct transportation system, shipments ranging in size from four pallets to full truck loads are picked up by a vehicle at the sender and are delivered by that same vehicle at the destination; there is no transshipment at any distribution center.

Operating the direct transportation system of Van Gend and Loos BV is a complex task, and management at Van Gend and Loos BV has realized that to handle the anticipated growth and to take advantage of the anticipated technological changes, such as global positioning systems and direct communication with vehicles, their planning department might benefit from more advanced planning tools than they currently have.

This paper discusses such a planning tool: DRIVE, Dynamic Routing of Independent VEhicles. DRIVE is to be incorporated in a decision support system for the direct transportation system at Van Gend and Loos BV. The heart of DRIVE is a branch-and-price algorithm for the general pickup and delivery problem (GPDP).

In a GPDP a set of routes has to be constructed in order to satisfy transportation requests. Each transportation request specifies the size of the load to be transported, the location where it is to be picked up (the origin), the location where it is to be delivered (the destination), and time windows defining allowable pickup and delivery times. Each load has to be transported by one vehicle from its origin to its destination without any transshipment at other locations. A heterogeneous fleet of vehicles is available to operate the routes. Each vehicle is characterized by a capacity, a depot where it is stationed, and a time window specifying when the vehicle is available.

The GPDP can naturally be formulated as a set partitioning problem in which the set of transportation requests has to be partitioned into a set of feasible routes. In recent years, set partitioning formulations have become very popular for many combinatorial optimization problems. There are two main reasons for this. First, for many problems alternative formulations are not known. This is the case, for example, in crew pairing problems (Anbil et al. 1993). Second, for many problems where alternative formulations are known, the linear programming relaxation of the set partitioning formulation often yields a stronger bound. This is the case, for example, in cutting stock problems (Vance et al. 1992).

Many large-scale set partitioning problems have been solved successfully by branch-and-price algorithms. Branch-and-price algorithms (Barnhart et al. 1998) solve mixed integer programming formulations with huge numbers of variables. In branch-and-price algorithms, sets of columns are left out of the linear program because there are too many columns to handle efficiently, and most of

Subject classifications: Transportation, vehicle routing: a dynamic pickup and delivery problem. Programming, integer, heuristic: a branch-and-price algorithm.
Area of review: COMPUTING AND DECISION TECHNOLOGY.

0030-364X/98/4604-0474 $05.00

them have their associated variable equal to zero in an optimal solution anyway. In order to check the optimality of a linear programming solution, a subproblem, called the *pricing problem*, is then solved in order to identify columns to enter the basis. If such columns are found, the linear program is reoptimized. Branching occurs when no columns price out to enter the basis, and the linear programming solution is optimal and fractional. Branch-and-price, which is a generalization of branch-and-bound with linear programming relaxations, allows column generation to be applied throughout the branch-and-bound tree.

Our discussion of DRIVE can be broken down into three parts. First, we present a branch-and-price algorithm for the GPDP. This algorithm is very flexible and is easily turned into an approximation algorithm that is capable of producing high quality solutions for large instances in an acceptable amount of computation time. This is a nontrivial task, since the underlying solution paradigm is based on the enumeration of the solution space, but essential because the algorithm is to be used in a real-life operational planning environment. The proper balance between quality and speed has been realized by using heuristics, whenever appropriate, within the overall optimization framework, by using a sophisticated column management scheme and problem specific information whenever possible in the branching scheme, the primal heuristic, and the column selection procedure. Secondly, we present the algorithmic adjustments that had to be made to the branch-and-price algorithm for the GPDP to be able to handle the specific characteristics of the direct transportation system of Van Gend and Loos BV. Finally, we present the results of a case study. These results show the viability of our approach and illustrate that sophisticated optimization based heuristics can be used successfully in complex real-life situations. This is one of the contributions of our work, since there seems to be a tendency to resort to relatively simple heuristics to solve complex real-life decision problems.

As the results of the case study were very encouraging, the management at Van Gend and Loos BV has put forward a proposal to the board of directors that asks for permission and funds to build a decision support system for the direct transportation system with DRIVE as the core planning tool.

This paper is organized as follows. In Section 1, we discuss the problem characteristics of the direct transportation system at Van Gend and Loos BV, their current planning methodology, and the envisioned planning methodology. In Section 2, we present the branch-and-price algorithm we have developed for the GPDP. In Section 3, we discuss the algorithmic adjustments that had to be made to the branch-and-price algorithm to be able to handle the specific characteristics of the direct transportation system of Van Gend and Loos BV. In Section 4, we present the computational experiments that have been conducted. Finally, in Section 5, we make some concluding remarks.

1. THE DIRECT TRANSPORTATION SYSTEM

In this section, we take a closer look at the direct transportation system at Van Gend and Loos BV. We discuss problem characteristics as well as current and envisioned solution methodology.

1.1. Problem Characteristics

1.1.1. Requests. All requests specify one pickup location and one or more delivery locations, which have to be visited in a predefined order. A single origin-destination pair is called a *shipment*. A request consisting of multiple delivery locations is said to consist of multiple shipments. These shipments have to be picked up at the same time by a single vehicle. All requests specify time windows for the pickup and all the deliveries.

Not all the transportation requests are known in advance; some become available in real time. This implies that at the time a new request becomes available, the set of routes that is currently being executed has to be modified. At the beginning of a day, about 60 percent of all requests that have to be served during that day are known.

1.1.2. Vehicles. There is a heterogeneous fleet of vehicles. Vehicle capacities are specified in pallets; there are vehicles with a capacity of 14, 16, 24, and 26 pallets. Capacity is not the only characteristic that is used to differentiate between vehicles. Vehicles are also distinguished based on their physical characteristics, because some clients have specific demands on the properties of a vehicle that serves their request.

The fleet of vehicles is not stationed at a central depot; each vehicle has its own *home location*.

A *working period* of a vehicle is a period of consecutive days in which the vehicle is in use. A vehicle starts and ends a working period at its home location with no load on board. Each night of a working period a driver sleeps at one of a set of *sleeping locations*, which includes his own home location. On Friday night a vehicle has to return home. At this time, the vehicle does not have to be empty, meaning that its working period has not yet ended. On Monday morning the vehicles are again available at their home locations. A driver must have a 45-minute lunch break between 11 a.m. and 2 p.m. each day of a working period.

The vehicles are rented by Van Gend and Loos BV on a daily basis for working periods of unknown length. When the number of transportation requests decreases, some vehicles will be sent home in order to end their working period. A number of vehicles are rented permanently, i.e., they cannot be returned. When the number of transportation requests increases, new vehicles are rented. A new vehicle can start its working period only at the beginning of the next working day. This implies that at the end of a day, when only part of the requests that have to be served the next day are known, the vehicles for the next day have to be selected. Van Gend and Loos BV does not exchange one vehicle for another, but returns vehicles only when the

total number of vehicles that is currently in use exceeds the number of vehicles that will be needed the next day.

1.1.3. Costs. The costs incurred by serving the requests are the drivers' pays, which include the lease costs of the vehicles. Each day of a working period a driver gets paid an amount proportional to the distance traveled that day unless this amount does not exceed some specified minimum, in which case this minimum is paid. Furthermore, a driver gets a compensation for each night of a working period that he does not spend at his home location.

1.1.4. Problem Size. The planning area, i.e., the region in which all locations are situated, is the Benelux. About 100 vehicles are used per day, of which 50 vehicles are rented permanently. The number of requests that has to be served per day is about 250 to 300. The total number of shipments per day is about 500.

1.2. Current Methodology

A central dispatching office collects new requests and assigns them to vehicles. At every stop of a route the driver of a vehicle calls this central office for new instructions. At present, the office cannot directly contact the drivers, although it is anticipated that a more sophisticated communication system will be implemented in the future.

The planning is done by a team of five people. Two of them do the actual planning, and the other three act as an interface between the planners and the drivers. All five people take turns on both jobs. The planning area is divided into two parts, each part covered by one planner. The planners basically use two sources of information: a *request list* with all transportation requests that have to be served within the next 20 hours, and a *vehicle list* with information about where and when the vehicles that are in use will become empty. On both lists, the items are geographically grouped. On the request list, this grouping is done by considering the origins of the requests. Within a group, the requests are listed in order of nondecreasing earliest pickup time. On the vehicle list, vehicles within one group are listed in order of nondecreasing time when they become empty.

The planners use a three-phase approach. First, they try to find combinations of requests that should be served by one vehicle. Requests are combined, based on proximity of origins and destinations in both space and time, and based on total load. In the second phase, these combinations are tentatively assigned to vehicles. Only when a driver calls for new instructions are these tentative assignments made permanent.

During the first part of a day the planners focus only on the work that has to be done during that day. In the afternoon, the following day is also considered. At this time the planners must also decide how many vehicles they need for the next day. Though only a fraction of the requests of the next day are known at that time, usually there is some knowledge about the total amount of work that has to be done during the next day. Such knowledge is available as planner's expertise. The number of vehicles that will be used the next day is primarily based on the amount of work that has to be done during the first part of the morning, but is increased if the expected total amount of work for the next day makes this necessary.

1.3. Envisioned Methodology

Because the pickup and delivery problem at Van Gend and Loos BV is a dynamic problem, we designed DRIVE to work in an iterative way. When DRIVE is invoked, it produces a plan, based on the current set of routes, the current set of known requests, and some estimate on future workload. This plan should be seen as a base plan that a planner can modify. While evaluating the base plan, a planner must focus primarily on the short-term decisions proposed by DRIVE, i.e., the assignments of loads to vehicles, and the routing of the vehicles within, say, the next hour. As soon as the plan is accepted, these short-term decisions are made permanent. This means that the first parts of the routes in the plan will be executed as planned. The remaining parts may be changed when new requests become available. As soon as necessary, DRIVE is used again to produce a new plan that respects all permanent decisions.

In this way, all routes are divided into a *head*, which is the part that will be executed as planned, and a *tail*, which is the remaining part that might change in the future. In this environment a planner is always busy preparing the tails of the routes. When a driver calls for new instructions, no calculations have to be performed because these instructions have been previously stored as the head of the driver's route.

2. A BRANCH-AND-PRICE ALGORITHM FOR THE GPDP

The core structure of the direct transportation system at Van Gend and Loos BV is a GPDP. For a survey of the GPDP see Savelsbergh and Sol (1995). Although the GPDP does not capture all the characteristics of the direct transportation system at Van Gend and Loos BV, we felt that an efficient and effective algorithm for the general pickup and delivery problem would provide an invaluable tool.

We decided to develop a set partitioning based algorithm for the GPDP. The primary reasons for this decision were:

- A set partitioning approach focuses on the assignment of transportation requests to vehicles, which is the most important and most difficult aspect of multivehicle routing problems with relatively few stops per trip.
- A set partitioning approach is very flexible in the sense that most restrictions imposed on the routes can easily be incorporated.

- Set partitioning approaches have provided very promising results for various types of vehicle routing problems (Dumas et al. 1991, Desrochers et al. 1992).
- Set partitioning approaches benefit from the rapid advances in linear programming technology.

Our efforts have resulted in the branch-and-price algorithm for the GPDP described below. This algorithm forms the basis of DRIVE.

2.1. A Set Partitioning Formulation

Let N be the set of transportation requests. For each transportation request $i \in N$, a load of size $q_i \in \mathbf{N}$ has to be transported from origin i^+ to destination i^-. Define $N^+ := \{i^+ | i \in N\}$ as the set of origins and $N^- := \{i^- | i \in N\}$ as the set of destinations. For each request $i \in N$ the pickup time window is denoted by $[e_{i^+}, l_{i^+}]$ and the delivery time window by $[e_{i^-}, l_{i^-}]$. Furthermore, let M be the set of vehicles. Each vehicle $k \in M$ has a capacity $Q_k \in \mathbf{N}$, is available in the interval $[e_{k^+}, l_{k^+}]$, and is stationed at depot k^+. Define $M^+ := \{k^+ | k \in M\}$ as the set of depots. Let $V := N^+ \cup N^- \cup M^+$.

For all $i, j \in V$, $k \in M$ let d_{ij} denote the travel distance, t_{ij}^k the travel time, and c_{ij}^k the travel cost. Note that the dwell time at origins and destinations can be easily incorporated in the travel time and therefore will not be considered explicitly.

Definition 1. A pickup and delivery route R_k for a vehicle of type k is a directed cycle through a subset $V_k \subset N^+ \cup N^- \cup \{k^+\}$ such that:

1. R_k starts in k^+ not before e_{k^+}.
2. For all $i \in N$, $i^+ \in V_k$ if and only if $i^- \in V_k$.
3. If $\{i^+, i^-\} \subseteq V_k$, then i^+ is visited before i^-.
4. Each location in $V_k \backslash \{k^+\}$ is visited exactly once.
5. Each location in $V_k \backslash \{k^+\}$ is visited within its time window.
6. The vehicle load never exceeds Q_k.
7. R_k ends in k^+ not after l_{k^+}.

To formulate the pickup and delivery problem as a set partitioning problem, we define

$\Omega_k :=$ the set of all feasible pickup and delivery routes for vehicle k,

$$\delta_{ir}^k := \begin{cases} 1 & \text{if } i \in N \text{ is served on route } r \in \Omega_k \\ 0 & \text{otherwise,} \end{cases}$$

$c_r^k :=$ the cost of route $r \in \Omega_k$,

and introduce binary variables $x_r^k (k \in M, r \in \Omega_k)$ equal to 1 if route $r \in \Omega_k$ is used and 0 otherwise. The pickup and delivery problem can now be formulated as follows:

$$\text{minimize} \sum_{k \in M} \sum_{r \in \Omega_k} c_r^k x_r^k,$$

$$\sum_{k \in M} \sum_{r \in \Omega_k} \delta_{ir}^k x_r^k = 1 \quad \text{for all } i \in N, \text{ (partitioning constraints),}$$

$$\text{subject to} \sum_{r \in \Omega_k} x_r^k \leqslant 1 \quad \text{for all } k \in M, \text{ (availability constraints),}$$

$$x_r^k \in \{0, 1\} \quad \text{for all } k \in M, r \in \Omega_k.$$

We denote this formulation by P and its linear programming relaxation by LP.

We will consider pickup and delivery problems where the primary objective is to minimize the number of vehicles needed to serve all transportation requests and the secondary objective is to minimize the total distance traveled. This is accomplished by taking the objective function

$$\sum_{k \in M} \sum_{r \in \Omega_k} (F + L_r^k) x_r^k,$$

where L_r^k denotes the length of route r for vehicle k and where $F > |N| \max_{r \in \Omega_k, k \in M} L_r^k$ is a large constant. This cost structure can be achieved by defining the travel costs $c_{ij}^k = d_{ij} (i \neq k^+)$, and $c_{k^+j}^k = F + d_{k^+j} (j \neq k^+)$.

2.2. A Branch-and-Price Algorithm

We have developed a branch-and-price algorithm for the GPDP based on the formulation presented above. The lower bound provided by LP is usually excellent and often much better than the lower bounds provided by more traditional formulations with variables x_{ij}^k indicating whether or not a vehicle of type k travels from location $i \in V$ to location $j \in V$.

A column generation scheme has been applied to solve LP in order to handle the large number of variables that arises due to the size of the sets Ω_k. Instead of explicitly enumerating all feasible routes in order to find a variable that prices out to enter the basis, in a column generation approach the nonbasic variable with the smallest negative reduced cost is found by solving an optimization problem, called the *pricing problem*. In this way the feasible routes are generated on the fly as needed, and only a small fraction of all feasible routes are used to solve LP.

Dumas et al. (1991) were the first to develop and implement a branch-and-price algorithm for the pickup and delivery problem with time windows. Our branch-and-price algorithm differs from theirs in various aspects. Most of these differences are prompted by the necessity to be able to solve large instances quickly.

The key features to accomplish a proper balance between solution speed and solution quality are:

- The use of heuristics, whenever possible, to solve the pricing problem.
- The use of a sophisticated column management scheme to keep the linear programs as small as possible and to keep the number of linear programs that have to be solved as small as possible.

- The use of a column selection procedure that increases the chances of finding or constructing feasible solutions within the active set of columns.
- The use of a branching scheme that concentrates on high-level decisions.
- The use of linear programming based primal heuristics.

2.2.1. Solving the Linear Programming Relaxation. Suppose that for each vehicle $k \in M$ a set $\Omega'_k \subseteq \Omega_k$ of feasible pickup and delivery routes is explicitly known. The *restricted master problem LP'* is defined as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{k \in M} \sum_{r \in \Omega'_k} c_r^k x_r^k, \\
& \sum_{k \in M} \sum_{r \in \Omega'_k} \delta_{ir}^k x_r^k = 1 && \text{for all } i \in N, \\
\text{subject to} \quad & \sum_{r \in \Omega'_k} x_r^k \leqslant 1 && \text{for all } k \in M, \\
& x_r^k \geqslant 0 && \text{for all } k \in M,\ r \in \Omega'_k.
\end{aligned}$$

Suppose that LP' has a feasible solution x, and let (u, v) be the associated dual solution, i.e., the dual variables $u_i (i \in N)$ are associated with the partitioning constraints and the dual variables $v_k (k \in M)$ are associated with the availability constraints. From linear programming duality we know that x is optimal with respect to LP if and only if for each $k \in M$ and for each $r \in \Omega_k$ the reduced cost d_r^k is nonnegative, i.e.,

$$d_r^k = c_r^k - \sum_{i \in N} \delta_{ir}^k u_i - v_k \geqslant 0 \quad \text{for all } k \in M,\ r \in \Omega_k.$$

Testing the optimality of x with respect to LP can thus be done by solving the *pricing problem*

$$\min\left\{ c_r^k - \sum_{i \in N} \delta_{ir}^k u_i - v_k \,\middle|\, k \in M,\ r \in \Omega_k \right\}.$$

Let z denote the value of the solution to the pricing problem, and let k_z and r_z denote the corresponding vehicle type and route. If $z \geqslant 0$, then x is also optimal with respect to LP, otherwise r_z defines a column that can enter the basis and has to be added to Ω'_{k_z}. This yields the following *column generation scheme*:

Step 1. Find initial sets Ω'_k containing a feasible solution x.

Step 2. Solve the restricted master problem LP'.

Step 3. Solve the pricing problem. If $z \geqslant 0$ then stop, otherwise set $\Omega'_{k_z} := \Omega'_{k_z} \cup \{r_z\}$ and go to Step 2.

Due to the presence of the availability constraints, it is nontrivial to find initial sets $\Omega'_k \subset \Omega_k$ containing a feasible solution to LP. However, if they exist, such sets can always be found using a two-phase method similar in spirit to the two-phase method incorporated in simplex algorithms to find an initial basic feasible solution. Define LP_+ as

$$\begin{aligned}
\text{minimize} \quad & \sum_{k \in M} \sum_{r \in \Omega_k} c_r^k x_r^k + \sum_{i \in N} p y_i, \\
& \sum_{k \in M} \sum_{r \in \Omega_k} \delta_{ir}^k x_r^k + y_i = 1 && \text{for all } i \in N, \\
\text{subject to} \quad & \sum_{r \in \Omega_k} x_r^k \leqslant 1 && \text{for all } k \in M, \\
& x_r^k \geqslant 0 && \text{for all } k \in M,\ r \in \Omega_k, \\
& y_i \geqslant 0 && \text{for all } i \in N,
\end{aligned}$$

where $y_i (i \in N)$ is an artificial variable and $p > \max_{k \in M, r \in \Omega_k} c_r^k$ is an appropriate penalty cost. Problem LP_+ can be solved by the above column generation scheme by initializing $\Omega'_k = \emptyset$ for each $k \in M$.

The artificial variables y_i are not deleted when they have all become nonbasic, i.e., when a feasible solution to problem LP has been found. Because of their high cost, these variables will stay nonbasic and will not interfere in the optimization process. However, during the branching process, the sets Ω'_k are restricted by the branching scheme, possibly yielding an initial infeasible LP in a node. In that case, the artificial variables will reappear in the basis, and the first phase is automatically started in order to find sets Ω'_k that do contain a feasible solution for the linear program associated with this node.

Because the route costs $c_r^k = F + L_r^k$ are constructed such that $F > |N| \max_{r \in \Omega_k, k \in M} L_r^k$, we can improve the lower bound Z_{LP} when the optimal LP solution x corresponds to a nonintegral number of vehicles. More precisely, if $m = \lceil \sum_{k \in M} \sum_{r \in \Omega_k} x_r^k \rceil$, then the constraint

$$\sum_{k \in M} \sum_{r \in \Omega_k} x_r^k \geqslant m,$$

is a valid inequality that may be added to LP, see also Desrosiers et al. (1996). Adding this constraint does not change the pricing problems, because the dual value of this constraint appears as a constant in their objective functions.

The column generation scheme presented above can be made more flexible based on the two following observations:

- Any column with negative reduced cost is a candidate to enter the basis and can be added to the restricted master.
- If several columns with negative reduced cost exist, they can be added simultaneously to the restricted master.

This flexibility can be exploited effectively to improve the overall efficiency of a column generation scheme. The basic idea is to solve the pricing problem approximately as long as this produces columns with a negative reduced cost, and solve the pricing problem optimally only when solving it approximately fails to produce columns with a negative reduced cost. In this way, the number of times the pricing problem is solved optimally (which is computationally prohibitive) is reduced considerably.

We have embedded approximation as well as optimization algorithms for the pricing problem into an effective and efficient column management system. The approximation algorithms try to generate many routes with negative reduced cost very fast. Any route with a negative reduced cost, whether generated by an approximation algorithm or an optimization algorithm, is stored in a *column pool*. Rather than solving the pricing problem at every iteration, we first search the column pool for columns with negative reduced cost. If successful, one or more of these are selected and added to the restricted master problem. If unsuccessful, we clean the column pool and invoke the pricing algorithms in order to try to refill the pool. Note that each time the restricted master problem is reoptimized the dual variables change. Therefore, the reduced costs of the columns in the pool have to be updated after every reoptimization. Cleaning the pool consists of removing all columns with reduced cost larger than some threshold $D_{\max} \geqslant 0$. A positive threshold value can be useful, because the reduced costs change after every reoptimization, possibly to a negative value. If the pricing problem is solved approximately, *LP* cannot be solved to optimality, so *P* cannot be solved to optimality. Therefore, as soon as the approximation algorithms fail to produce columns with negative reduced cost, an optimization algorithm is used to solve the pricing problem optimally to either prove optimality of *LP* or find new columns with negative reduced costs. The column generation scheme we propose now looks as follows.

Step 1. Find initial sets Ω_k' containing a feasible solution x.

Step 2. Set the column pool equal to $\emptyset$.

Step 3. Solve the restricted master problem LP'.

Step 4. If the column pool contains columns with negative reduced cost, select some of these columns, add them to the restricted master problem, and go to Step 3.

Step 5. Delete columns with reduced cost larger than $D_{\max}$ from the column pool and start the approximation algorithms for the pricing problem. If these are successful, add the generated columns to the pool and go to Step 4.

Step 6. Solve the pricing problem to optimality. If $z \geqslant 0$, then stop; otherwise, add the generated columns to the pool and go to Step 4.

There are several ways to choose columns from the column pool to add to the restricted master problem. The first possibility is to select the column with the minimum reduced cost. In this way, the linear program will not grow very rapidly, but a linear program has to be solved for each added column. A second possibility is to select all columns with negative reduced costs from the pool. This will reduce the number of linear programs that have to be solved, but the linear programs will become very large. We have chosen a more adaptive greedy selection scheme that selects partial solutions to problem *P*. More precisely, we select a set of columns with negative reduced costs that correspond to a set of routes satisfying the following requirements:

- Each transportation request is served on at most one route.
- The number of vehicles of a certain type assigned to the routes does not exceed the available number of vehicles of that type.

The set of columns is constructed by successively choosing a column with minimum negative reduced cost such that the two requirements are still satisfied. The selection stops when no more such columns are available in the column pool.

The above column selection mechanism is motivated by two observations. First, adding columns corresponding to partial solutions to *P* increases the chance of encountering integral solutions during the solution of the master problem. Second, it prevents the addition of similar columns, which would happen if the columns would be selected merely based on their reduced cost and the dual variables are far from being optimal.

2.2.2. The Pricing Problem. The pricing problem decomposes into several independent problems, one for each vehicle, since

$$z = \min\left\{ c_r^k - \sum_{i \in N} \delta_{ir}^k u_i - v_k \,\middle|\, k \in M,\ r \in \Omega_k \right\},$$

is equal to

$$z = \min_{k \in M} \min\left\{ c_r^k - \sum_{i \in N} \delta_{ir}^k u_i - v_k \,\middle|\, r \in \Omega_k \right\},$$

i.e., the problem of finding a minimum cost route for vehicle k, using a modified cost structure, that serves a subset of the transportation requests. Denote these independent problems by S_k for $k \in M$. Problem S_k can be viewed as a shortest path problem with precedence constraints, capacity constraints, and time windows on the perturbed cost matrix c'.

A dynamic programming algorithm using labeling techniques to handle the precedence, capacity, and time constraints can be used to solve this shortest path problem (Dumas et al. 1991). However, for instances with many transportation requests and time and capacity constraints that are not very tight, solving S_k to optimality becomes computationally prohibitive.

As pointed out in the previous section, the pricing problem can be solved approximately as long as columns with negative reduced cost are found. Solving the pricing problem approximately can be done in several ways.

One approach is to speed up the dynamic programming algorithm in earlier iterations of the column generation process by working on a reduced network (Dumas et al. 1991). Network reduction is achieved by deleting nodes corresponding to requests with a low dual value and by deleting arcs with relatively high cost. Obviously, this reduces the state space of the dynamic program and thus the

computation times, but it no longer guarantees that the optimal solution is found. If no more profitable routes can be found in the reduced network, it is enlarged and the dynamic programming algorithm is started again. Note that this approach guarantees that in the end the pricing problem is solved to optimality.

Also note that the dynamic programming algorithm may encounter many columns with negative reduced costs before it identifies the one with the smallest reduced cost. Obviously, all these columns could be stored in the column pool. However, since the number of columns with negative reduced cost in the first iterations of the column generation process is huge, this would lead to an unmanageable column pool. Furthermore, only a few of the columns generated in the first iterations of the column generation scheme will be actually added to the restricted master problem. Therefore, in our implementation we have put an upper bound on the number of columns that the algorithm can create in one run for each vehicle. The algorithm will stop as soon as the upper bound is reached, which reduces computation times drastically. Note that this approach still guarantees that in the end the pricing problem is solved to optimality.

Another approach is to use fast approximation algorithms instead of the dynamic programming algorithm. The fast approximation algorithms we have used are based on construction and improvement algorithms for the single-vehicle GPDP. In their description, we use $I_r^k(j)$ to denote the minimal cost of inserting transportation request j into route r for vehicle k. If transportation request j cannot be inserted into route r, then $I_r^k(j) = \infty$.

Since computing the true insertion cost involves the solution of a single-vehicle pickup and delivery problem with time windows, i.e., a traveling salesman problem with time windows, precedence constraints, and capacity constraints, we have chosen to work with an approximate insertion cost, namely the cheapest insertion cost. If r is a route consisting of n stops, our insertion algorithm first calculates τ_i $(1 \leq i \leq n)$, which is the latest possible time the vehicle may arrive at stop i in order to ensure feasibility, with respect to the time windows of the remaining part of the route $(i, i+1, \ldots, n)$. Because a request requires two stops to be inserted into a route, there are $O(n^2)$ possible insertions for each request. By evaluating these insertions in the right order and using the values τ_i, each possible insertion can be checked for feasibility and cost in constant time. The cheapest insertion cost of request j into route r can therefore be computed in $O(n^2)$ time.

Construction Algorithms. Construction algorithms build routes from scratch. Define the reduced insertion cost $D_r^k(j) := I_r^k(j) - u_j$. The construction algorithms initialize a route r and then repeatedly try to decrease the reduced cost of the route by inserting a request with $D_r^k(j) < 0$. If $D_r^k(j) \geq 0$ for all requests not in the route, and $d_r^k < 0$, then r is added to the column pool. The route can be initialized as a loop from k^+ to k^+, or as a route serving some requests that are likely to be served by a vehicle of type k.

Improvement Algorithms. Improvement algorithms modify existing routes. Note that the current *LP* solution provides us with a set of routes r with $d_r^k = 0$ (at least all the routes associated with basic variables). When such a route is used as a starting point for a local search algorithm, we expect to find a route with negative reduced cost very fast. The following two algorithms start with a route r with $d_r^k = 0$ and then try to decrease the reduced cost of the route by deleting requests from the route and replacing them by other requests. The first algorithm performs profitable swaps until no such swap can be found. The second algorithm performs a variable depth search. When $i \in N$ is served on route r, we denote by $r\backslash\{i\}$ the route obtained from r by deleting request i. When $j \in N$ is not served on route r, we denote by $r \cup \{j\}$ the route obtained from r by inserting request j in the cheapest way.

The first algorithm works as follows.

Step 1. Let $d^k_{(r\backslash\{i_0\})\cup\{j_0\}} = \min\{d^k_{(r\backslash\{i\})\cup\{j\}} | i, j \in N, \delta^k_{ir} = 1, \delta^k_{jr} = 0\}$.

Step 2. If $d^k_{(r\backslash\{i_0\})\cup\{j_0\}} < d_r^k$, then set $r = (r\backslash\{i_0\}) \cup \{j_0\}$ and go to Step 1.

Step 3. If $d_r^k < 0$, then add r to the column pool.

At each iteration of the variable depth search algorithm the best swap is performed, even if this increases the reduced cost. The algorithm maintains a set $F \subset N$ of requests that were deleted from the route in a previous iteration. These requests are not allowed to reenter the route. The best route found over all iterations is added to the column pool if it has negative reduced cost:

Step 1. $\bar{r} = \emptyset$, $d_{\bar{r}}^k = \infty$, and $F^k = \emptyset$.

Step 2. Let $d^k_{(r\backslash\{i_0\})\cup\{j_0\}} = \min\{d^k_{(r\backslash\{i\})\cup\{j\}} | i, j \in N, \delta^k_{ir} = 1, \delta^k_{jr} = 0, j \notin F^k\}$.

Step 3. If $d^k_{(r\backslash\{i_0\})\cup\{j_0\}} = \infty$, then go to Step 6, otherwise set $r = (r\backslash\{i_0\}) \cup \{j_0\}$ and $F^k = F^k \cup \{i_0\}$.

Step 4. If $d_r^k < d_{\bar{r}}^k$, then set $d_{\bar{r}}^k = d_r^k$ and $\bar{r} = r$.

Step 5. Go to Step 2.

Step 6. If $d_{\bar{r}}^k < 0$, then add $\bar{r}$ to the column pool.

2.2.3. Solving the Integer Program. In order to obtain integral solutions, we need a branching scheme that excludes the current fractional solution, validly partitions the solution space of the problem, and does not complicate the pricing problem too much. The third requirement almost always excludes the standard branching rules based on variable fixing. Fixing a variable to 1 does not complicate the pricing problem, because it just reduces the size of the problem. However, fixing a variable to 0 corresponds to forbidding a certain solution to the pricing problem. Deeper down the search tree this implies that a set of

solutions to the pricing problem must be excluded, which is in general very complicated, if not impossible.

In order to develop branching schemes that satisfy the requirements given above, the structure of the pickup and delivery problem has to be exploited. The pickup and delivery problem can be roughly divided into two parts. First, the assignment of the requests to the available vehicles. Second, the construction of pickup and delivery routes. This suggests two types of branching schemes: one that partitions the solution space with respect to assignment decisions, the other with respect to routing decisions.

Branching on Routing Decisions. Dumas et al. (1991) present a branching scheme that focuses on routing decisions. The branching scheme is based on the one proposed for the asymmetric traveling salesman problem by Carpaneto and Toth (1980). Let $x_r^k > 0$ be a fractional variable in the current LP solution. Suppose that the corresponding route r serves n requests $\{i_1, i_2, \ldots, i_n\} \subset N$ in such a way that i_p is picked up before i_q if $p < q$. Binary order variables O_{ij} $(i, j \in N^+ \cup M^+)$ are introduced, where O_{ij} is equal to 1 if no requests are picked up between locations i and j, and 0 otherwise. The current subset of solutions is now divided into $n + 2$ subsets as follows. Define $i_0^+ = i_{n+1}^+ = k^+$. The first subset is characterized by the constraints $O_{i_0^+ i_1^+} = O_{i_1^+ i_2^+} = \cdots = O_{i_n^+ i_{n+1}^+} = 1$. For each $j \in \{0, 1, \ldots, n\}$ another subset is characterized by $O_{i_0^+ i_1^+} = O_{i_1^+ i_2^+} = \cdots = O_{i_{j-1}^+ i_j^+} = 1$ and $O_{i_j^+ i_{j+1}^+} = 0$. The dynamic programming algorithm to solve the pricing problem can be easily modified such that the routes found by the algorithm satisfy the constraints on the order variables.

Branching on Assignment Decisions. We present an alternative branching scheme that focuses on assignment decisions rather than routing decisions. Assignment decisions constitute higher level decisions and have a greater impact on the structure of the solution. Therefore, we feel that assignment decisions are more important than routing decisions and should be made first.

Let x be the current fractional solution to *LP*. Now define for each $i \in N$ and $k \in M$ the *assignment value* $z_i^k = \sum_{r \in \Omega_k} \delta_{ir}^k x_r^k$, indicating what fraction of request i is served by vehicle k in the current LP solution.

Proposition 1. *Let x be an optimal solution of LP*, and let $z_i^k = \sum_{r \in \Omega_k} \delta_{ir}^k x_r^k$ $(k \in M, i \in N)$. Then x is integral if and only if z is integral.

Proof. If x is integral, then trivially z is integral. Now suppose z is integral. Let $k \in M$ and $i \in N$ satisfy $z_i^k = 1$. Because $\sum_{r \in \Omega_k} \delta_{ir}^k x_r^k = 1$ and $\sum_{r \in \Omega_k} x_r^k \leq 1$, we have that $\delta_{ir}^k = 1$ for all $r \in \Omega_k$ with $x_r^k > 0$. Therefore, all routes $r \in \Omega_k$ with $x_r^k > 0$ serve the same requests. Because x is an optimal solution to LP, these routes all have the same cost, so they are identical. Since all routes in Ω_k are distinct, this implies that x is an integral solution. □

Based on the above proposition, we propose the following branching scheme: *when x is fractional, find a request* $\hat{\imath} \in N$ *and a vehicle* $\hat{k} \in M$ *with* $0 < z_{\hat{\imath}}^{\hat{k}} < 1$, *and create two subsets characterized by* $z_{\hat{\imath}}^{\hat{k}} = 0$ *and* $z_{\hat{\imath}}^{\hat{k}} = 1$, *respectively.*

This branching scheme can be viewed as a special case of the branching scheme proposed by Ryan and Foster (1981) and does not complicate the pricing problem in the subsets. The restriction $z_{\hat{\imath}}^{\hat{k}} = 0$ is easily satisfied by ignoring request $\hat{\imath}$ when solving pricing problem $S_{\hat{k}}$. It is less obvious that the restriction $z_{\hat{\imath}}^{\hat{k}} = 1$, i.e., requiring that request $\hat{\imath}$ is served by vehicle $\hat{k}$, can also be satisfied when solving pricing problem $S_{\hat{k}}$. In this case, it seems to depend on the specific algorithm used. For example, if a construction heuristic is used it is easy and if a dynamic programming algorithm is used it is doable. Sol (1994) shows that, in fact, any algorithm for the pricing problem $S_{\hat{k}}$ can be used with an appropriate choice of dual variables.

2.2.4. Primal Solutions. In order to keep the search tree small, we need good lower and upper bounds. To obtain good upper bounds, we have developed a primal heuristic that, in each node of the search tree, tries to construct a feasible solution starting from the current fractional solution and, if successful, tries to improve this solution.

The constructive algorithm is based on the assignment values defined in the previous section. Let x be the fractional solution to the *LP*. Then we define for each request $i \in N$ and each vehicle $k \in M$ the fractional assignment value $z_i^k = \sum_{r \in \Omega_k} \delta_{ir}^k x_r^k$. Note that if this value is large, the LP solution indicates that it is likely that transportation request i is served by vehicle k. The following algorithm now tries to construct a feasible solution:

Step 1. $N_0 = N$.
Sort the pairs $(k, i) \in M \times N$ such that $z_{i_1}^{k_1} \geq z_{i_2}^{k_2} \geq z_{i_3}^{k_3} \geq \ldots$
$t = 1$.
For each vehicle $k \in M$ set r_k to be the empty route of k.

Step 2. If $i_t \notin N_0$ or i_t cannot be added to route r_{k_t}, then go to 4.

Step 3. Add i_t to route r_{k_t}.
Remove i_t from N_0.

Step 4. $t = t + 1$.
If $N_0 \neq \emptyset$ and $t \leq |M||N|$, then go to 2, otherwise stop.

Checking whether i_t can be added to route r_{k_t} in Step 2 is done with our cheapest insertion algorithm.

If a solution is found in this way, it is subjected to three local search algorithms. The first algorithm considers a single route and *reinserts* each request. Because the routes were constructed by sequential insertion, the cheapest insertion of a request in its route can differ from its current positions. The other two algorithms consider two routes and try to decrease the total cost by moving requests from one route to the other, or by exchanging two requests

between routes. The cost of both operations is approximated by insertion and deletion algorithms.

The quality of the upper bound may be further improved by incorporating more sophisticated iterative improvement algorithms, such as those described in Kindervater and Savelsbergh (1997).

2.3. Computational Experiments

The ultimate goal of our research is the development of a high-quality approximation algorithm for the GPDP that can solve moderate size instances in an acceptable amount of computation time. We believe that, with the appropriate choices, the branch-and-price algorithm described above satisfies these requirements, and the results of the various computational experiments that we have conducted support that claim.

The computational experiments are designed to answer the following questions.

- Do we solve the linear relaxations faster when we use the column management scheme described in Section 2.2.1?
- If we never solve the pricing problem to optimality, i.e., if we use the heuristics only to solve the pricing problem, can we still get high-quality solutions?
- Does generating columns at every node of the search tree, as opposed to generating columns only in the root node, lead to higher quality solutions?

We have implemented several versions of our algorithm to be able to answer these questions, and we start their description with a discussion of the implementation issues that are common to all of them.

The cheapest insertion algorithm that we have developed for the pricing problem is also used to construct a starting solution and an initial set of columns. The limit on the number of columns that the dynamic programming algorithm can generate for each vehicle in a single execution is set to 200. We have experimented with limits of 50, 200, and 1,000 columns per vehicle, but the computation times hardly varied. Because the limit of 200 columns per vehicle produced slightly better results, this value has been chosen. The threshold value used to decide when the column pool is cleaned up has been set to 10. The branching pair $(\hat{k}, \hat{\imath})$ is chosen such that $z_{\hat{\imath}}^{\hat{k}} = \max\{z_i^k | z_i^k < 1, i \in N, k \in M\}$. The search tree is explored according to a best bound search.

We have implemented three approximation algorithms, A_1, A_2, and A_3. All of them use the column generation scheme of Section 2.2.1. The algorithms differ either in the way they solve the *LP*s in the root node or in the way they solve the *LP*s in the other nodes of the search tree. Algorithm A_1 always uses the heuristics to solve the pricing problem and therefore never solves the pricing problem to optimality. In the root node, algorithm A_2 uses the heuristics to solve the pricing problem as long as they produce profitable columns, but reverts to the dynamic programming algorithm when the heuristics fail to produce profitable columns. Consequently, in the root node A_2 solves the *LP* to optimality and therefore produces a valid lower bound on the optimal solution value. In all the other nodes A_2 uses the heuristics only to solve the pricing problem. In the root node algorithm A_3 is identical to algorithm A_2, but algorithm A_3 does not generate new columns in any of the other nodes of the search tree. As a consequence, algorithm A_3 circumvents the complications introduced by generating columns throughout the search tree and can use standard branching rules based on variable dichotomy with the given set of columns.

We have implemented two optimization algorithms, O_1 and O_2. Both of them use the column generation scheme of Section 2.2.1, but O_1 always uses the dynamic programming algorithm for column generation, whereas O_2 uses the heuristics to solve the pricing problem as long as they produce profitable routes.

All versions have been implemented using MINTO, a Mixed INTeger Optimizer (Nemhauser et al. 1994). MINTO is a software system that solves mixed-integer linear programs by a branch-and-bound algorithm with linear programming relaxations. It also provides automatic constraint classification, preprocessing, primal heuristics and constraint generation. Moreover, the user can enrich the basic algorithm by providing a variety of specialized application routines that can customize MINTO to achieve maximum efficiency for a problem class. All our computational experiments have been conducted with MINTO 2.0/CPLEX 3.0 and have been run on an IBM/RS6000 model 550.

2.3.1. Test Problems. Because the ability to solve the pricing problem to optimality, and thus the ability to solve an instance to optimality, strongly depends on the size of the set of all feasible solutions, and this size strongly depends on the number of transportation requests that can be in a vehicle at the same time and the width of the pickup and delivery time windows, we have developed a random problem generator that allows us to vary these instance characteristics.

Instances are constructed as follows. Generate a set of 100 points randomly within a square of size 200×200. The distance between two points is the Euclidean distance. The travel time between two points is equal to the distance between these points. Origins (i^+), destinations (i^-), and vehicle home locations (k^+) are now chosen from this set of points. The load of a request is selected from an interval $[q^{\min}, q^{\max}]$. The capacity of all vehicles is equal to Q. The time windows of the requests are constructed in the following way: The planning period has length $L = 600$. Each window has width W. For each request i choose e_{i^+} randomly within the interval $[0, e_i^{\max}]$, where $e_i^{\max} = L - t_{i^+i^-}$. The time windows for request i are now calculated as $[e_{i^+}, e_{i^+} + W]$ for the pickup and $[e_{i^+} + t_{i^+i^-}, e_{i^+} + t_{i^+i^-} + W]$ for the delivery. A time unit can be interpreted as a minute. In this way the length of the diagonal of the

Table I
Problem Classes

Class	$\|N\|$	$\|M\|$	$q^{\min}$	$q^{\max}$	Q	W
A30	30	15	5	15	15	60
B30	30	15	5	20	20	60
C30	30	15	5	15	15	120
D30	30	15	5	20	20	120

square corresponds to approximately half a planning period. We choose the number of available vehicles $|M| = |N|/2$.

As indicated earlier, the objective is to minimize the number of vehicles used and the total distance traveled. We have taken the fixed cost F to be $F = 10{,}000$.

Table I lists the problem classes that we have used in our first experiments in order of anticipated increasing difficulty. We have randomly generated 10 instances in each problem class. Although the number of transportation requests is not large, it is large enough to enable us to analyze the characteristics of the different versions of the algorithms we have proposed.

We have also tried to obtain the problem instances used by Dumas et al. (1991), but unfortunately, these are no longer available.

2.3.2. Quality of the Lower Bound. We first consider the quality of the lower bound Z_{LP} obtained in the root of the branch-and-bound tree. Note that this includes the addition of the constraint $\Sigma_{k \in M} \Sigma_{r \in \Omega_k} x_r^k \geq m$ (see Section 2.2.1). For all instances, the optimal number of vehicles equals m. We therefore focus on the quality of the lower bound with respect to the total distance traveled. Table II shows the linear programming bound at the root (Z_{LP}), the value of the optimal solution (Z_{OPT}), and the integrality gap (as a percentage of the optimal value) with respect to distance traveled only: $100(Z_{OPT} - Z_{LP})/(Z_{OPT} - mF)$. For 17 out of 40 instances this gap equals 0, indicating that the problem was solved without any branching, and only for 7 out of 40 instances the gap exceeds 1 percent.

Based on this experiment, we conclude that the lower bound computed in the root node provides a good indication of the value of an optimal solution.

2.3.3. The Optimization Algorithms. To analyze the effect of using heuristics in the column generation scheme on the time required to solve the linear relaxations, we have solved the initial linear programming relaxation by both algorithm O_1 and O_2 for all instances in the problem classes A30, B30, C30, and D30. Table III shows the averages of the CPU time, the number of columns generated, and the number of columns added over the 10 instances in each class. Details for all instances can be found at http://www.informs.org/Pat/will/fill/this/in. The number of columns generated is the total number of columns that have been stored in the column pool during the solution process. The last column shows the quotient CPU(O_2)/CPU(O_1).

Algorithm O_2 clearly outperforms O_1. Over all 40 instances, we have observed an average decrease in computation time of 35 percent when using the approximation algorithms for the pricing problem. For problem class D30 the average computation time was almost halved. The number of columns in the optimal master problem never becomes very large, and this number does not differ drastically for O_1 and O_2. There is, however, a big difference in the number of generated columns. This is due to the fact that the dynamic programming algorithm stores all columns with negative reduced cost it encounters (up to 200 per vehicle per execution) in the column pool. Although this results in a larger pool size for O_1 than for O_2, the

Table II
Optimal Solutions and Integrality Gaps

Problem	Z_{opt}	Z_{LP}	gap	Problem	Z_{opt}	Z_{LP}	gap
A30.1	104,291	104,263.0	0.65	B30.1	124,655	124,655.0	0
A30.2	114,451	114,451.0	0	B30.2	94,341	94,332.0	0.21
A30.3	135,415	135,415.0	0	B30.3	124,882	124,872.0	0.21
A30.4	115,148	115,148.0	0	B30.4	104,431	104,431.0	0
A30.5	114,535	114,535.0	0	B30.5	94,434	94,403.0	0.70
A30.6	114,967	114,939.8	0.55	B30.6	104,264	104,264.0	0
A30.7	104,247	104,247.0	0	B30.7	104,263	104,262.0	0.02
A30.8	115,154	115,146.0	0.16	B30.8	135,332	135,332.0	0
A30.9	114,883	114,883.0	0	B30.9	145,466	145,466.0	0
A30.10	115,208	115,208.0	0	B30.10	114,685	114,685.0	0
C30.1	84,664	84,604.0	1.29	D30.1	94,460	94,440.5	0.44
C30.2	84,674	84,674.0	0	D30.2	73,855	73,832.0	0.60
C30.3	84,011	83,961.7	1.23	D30.3	84,737	84,737.0	0
C30.4	74,205	74,168.7	0.86	D30.4	84,765	84,765.0	0
C30.5	84,426	84,391.2	0.79	D30.5	84,593	84,498.5	2.06
C30.6	94,547	94,500.5	1.02	D30.6	84,257	84,247.5	0.22
C30.7	84,270	84,187.4	1.93	D30.7	84,302	84,278.5	0.55
C30.8	94,778	94,767.0	0.23	D30.8	74,293	74,097.0	4.57
C30.9	84,246	84,231.2	0.35	D30.9	84,110	84,110.0	0
C30.10	84,230	84,213.2	0.40	D30.10	74,499	74,402.5	2.14

Table III
Performance of Optimization Algorithms for Z_{LP}

	O_1			O_2			
Problem	CPU	Generated	Added	CPU	Generated	Added	
A30	21.56	4183.2	195.5	13.55	921.3	207.2	0.70
B30	22.85	4511.7	194.0	15.93	878.0	203.6	0.68
C30	215.63	10,693.0	282.5	135.44	1493.3	280.6	0.63
D30	241.02	9808.3	275.0	139.07	1396.7	270.2	0.58

larger pool size does not cause the differences in computation times. The differences in computation times can be fully attributed to the fact that O_1 uses only optimization algorithms to solve the pricing problem, whereas O_2 also uses approximation algorithms.

Based on the above observations, we have chosen algorithm O_2 to solve all the problem instances to optimality. Table IV shows the total CPU time and the number of nodes evaluated in the search tree.

The results presented in Table IV indicate that for loosely constrained instances the time required to solve an instance may vary significantly and in some cases may become prohibitively large.

2.3.4. The Approximation Algorithms. The computational experiments with the optimization algorithm have demonstrated that there can be significant differences in solution times even for instances in the same problem class. This type of behavior is generally unacceptable in practical planning situations. An analysis of the distribution of the total computation time over the various components of the branch-and-price algorithm revealed that most of the time was spent on optimally solving the pricing problems. This suggests that we might be able to reduce the computation times and improve the robustness by reducing the number of times the pricing problem is solved to optimality. This observation motivated the three approximation algorithms.

Table V shows the averages of the CPU time, the number of evaluated nodes, and the relative error $(Z_{BEST} - Z_{OPT})/(Z_{OPT} - mF)$ over the 10 instances in each problem class for the approximation algorithms A_1, A_2, and A_3.

Algorithm A_2 clearly outperforms the others with respect to quality of the solutions. It solved 36 out of 40 problems to optimality. For 17 of these problems, this is due to the fact that no branching was required, and for 19 problems A_2 found the optimal solution even though branching was required and the pricing problems were solved to optimality only in the root node. For the four problems that were not solved to optimality, the relative error was at most 0.33 percent. By comparing the results of A_2 and A_3, we conclude that it pays to use column generation during branch-and-bound. However, as the relative errors of A_3 are still small, it is clear that creating a good set of columns in the root is the most important issue in finding good approximate solutions.

Algorithm A_1, which never solves the pricing problem to optimality, outperforms the others with respect to speed. For all problems the optimal number of vehicles was obtained, and though only 9 out of 40 problems are solved to optimality, the average relative error over all 40 problems is only 1.35 percent. These observations indicate that A_1 might be a good algorithm for practical situations where problem sizes are bigger and time and capacity constraints are less restrictive. In fact, in such situations it might not be possible to use the other algorithms because of the computation times of the dynamic programming algorithm.

Larger Instances. To determine whether the conclusions drawn based on the experiments with instances with 30 transportation requests remain valid when larger instances are solved, we have generated several sets of larger instances and tested algorithms A_1 and A_2 on these sets. We

Table IV
Performance of Optimization Algorithm O_2

Class	A30		B30		C30		D30	
Nr	CPU	#N	CPU	#N	CPU	#N	CPU	#N
1	106.95	15	8.61	1	60.82	3	177.99	7
2	11.08	1	39.07	5	110.54	1	2,552.56	23
3	6.83	1	14.87	3	684.49	27	83.93	1
4	12.85	1	13.28	1	1,115.42	35	77.82	1
5	12.54	1	81.61	9	1,082.63	57	6,402.77	873
6	78.23	47	15.05	1	410.75	91	345.78	23
7	10.75	1	19.68	3	7,148.49	329	205.44	9
8	19.11	5	5.41	1	183.12	9	55,266.79	1,555
9	5.67	1	10.91	1	457.93	17	242.40	1
10	9.23	1	15.09	1	1,035.06	23	897.31	51
Avg.	27.32	7.4	22.36	2.6	1,228.93	59.2	6,625.28	254.4

Table V
Performance of Approximation Algorithms

	A_1			A_2			A_3		
Problem	CPU	#N	Error	CPU	#N	Err	CPU	#N	Error
A30	7.73	6.9	1.46	14.55	5.0	0.03	13.88	5.0	0.00
B30	6.39	2.6	1.27	16.37	2.6	0.05	16.20	4.4	0.02
C30	37.55	28.40	0.71	148.74	27.8	0.11	138.07	24.6	0.60
D30	46.98	52.5	1.96	164.14	55.0	0.08	142.06	24.4	0.18

did not include algorithm A_3 since its computational requirements do not differ substantially from those of algorithm A_2 and A_2 generally produces higher quality solutions. The characteristics of these problem classes are shown in Table VI.

The set DAR50 is a set of instances of the dial-a-ride problem, which is a well-known special case of the pickup and delivery problem in which loads represent people. In dial-a-ride problems the capacity restrictions are fairly loose.

As our ultimate goal is the development of a high-quality approximation algorithm for the GPDP that can solve moderate size instances in an acceptable amount of computation time, we have imposed an upper bound of 10 minutes of cpu time. When this bound causes the algorithm to stop, we take the best solution found so far. We have done this to mimic practical situations in which there exists a limit on the time available to construct a set of routes.

In the next three tables, we again present only averages; the details can be found at http://www.informs.org/Pat/will/fill/this/in.

The results for classes A50 and B50 are shown in Table VII, and the results for classes C50 and D50 are shown in Table VIII. For the more loosely constrained problem classes, we can clearly observe the computational disadvantages associated with solving the pricing problem to optimality, since algorithm A_2 terminated prematurely for all but 2 instances, i.e., it ran into the time limit of 10 minutes. In fact, for all these instances, the algorithm has not been able to completely evaluate the root node! On the other hand, algorithm A_1 terminated normally for most instances and produces significantly better solutions. To determine the effect of the time limit on the quality of the solutions, we have run both algorithms on the same set of instances with a time limit of 30 instead of 10 minutes. The results can also be found in Table VIII and indicate that allowing more time does result in better solutions.

Table VI
Larger and Less Restricted Problem Classes

Class	$\|N\|$	$\|M\|$	$q^{\min}$	$q^{\max}$	Q	W
A50	50	25	5	15	15	60
B50	50	25	5	20	20	60
C50	50	25	5	15	15	120
D50	50	25	5	20	20	120
DAR50	50	25	1	1	5	60

The results presented in Tables VII and VIII indicate that algorithm A_2 performs well on tightly constrained instances, but that computational requirements become a bottleneck for loosely constrained instances. Algorithm A_1, on the other hand, produces high-quality solutions in an acceptable amount of computation time regardless of the problem class.

The results presented in Table IX indicate that both algorithms are capable of producing high quality solutions for instances of the dial-a-ride problem.

3. DRIVE

Since the general pickup and delivery problem does not capture all the characteristics of the direct transportation system at Van Gend and Loos BV, the branch-and-price algorithm cannot be applied directly. In this section, we discuss the modeling tricks and the algorithmic adjustments that had to be made to be able to handle the specific characteristics of the direct transportation system at Van Gend and Loos BV.

3.1. Handling the Dynamics

DRIVE uses an iterative approach to handle the dynamics of the direct transportation system. At each iteration, a static general pickup and delivery problem, which we call the reoptimization problem, is solved. Each reoptimization problem, which is characterized by a set of vehicles, a set of transportation requests, and a set of initial routes, depends on the solution obtained in the previous iteration. In the following subsections, we describe how the reoptimization problem is constructed when DRIVE is invoked at time τ.

3.1.1. Planning Horizon. Sometimes, requests are known long before they can be served. It is not necessary to include such requests into the planning process as soon as they become available. We therefore introduce a planning horizon $H \geq 0$, and we consider only those requests

Table VII
Performance of Algorithms $A1$ and $A2$ for the Problem Classes A50 and B50

	A_1			A_2		
Problem	Z_{BEST}	#N	CPU	Z_{BEST}	#N	CPU
A50	163,645.3	33.8	103.65	163,564.9	48.0	248.47
B50	167,526.5	3.1	30.95	167,416.7	11.2	167.32

Table VIII
Performance of Algorithms $A1$ and $A2$ for the Problem Classes C50 and D50

	A_1 (CPU $\leq$ 600)			A_1 (CPU $\leq$ 1800)		
Problem	Z_{BEST}	#N	CPU	Z_{BEST}	#N	CPU
C50	134,992.2	66.1	318.51	129,931.7	157.6	553.21
D50	132,749.3	121.7	453.22	123,683.1	363.4	758.99
	A_2 (CPU $\leq$ 600)			A_2 (CPU $\leq$ 1800)		
Problem	Z_{BEST}	#N	CPU	Z_{BEST}	#N	CPU
C50	164,470.9	4.0	592.81	156,318.3	86.9	1,539.85
D50	166,141.7	1.0	600.00	158,002.7	48.9	1,766.73

that can be picked up before time $\tau + H$. The parameter H also provides a means to control the size of the problem instance.

3.1.2. Requests. Let the active set of requests consist of all requests that are known at time τ that have not yet been completed, and that can be picked up before time $\tau + H$. We distinguish two types of active requests: permanently assigned requests and non-assigned requests. An active request that is served on an existing route and that has been picked up before τ is labeled as a permanently assigned request, since it has to be assigned to the vehicle associated with the route.

For each existing route, we introduce a virtual request representing all permanently assigned (but not yet completed) requests, i.e., representing the loads that are on board the vehicle at τ. The origin of the virtual request is the first location on the existing route visited after time τ and the destinations of the virtual request are the locations where the loads that are on board the vehicle must be delivered. The set of virtual requests replaces the set of permanently assigned requests. For the virtual requests we relax the constraint that deliveries have to be made in a predefined order, because the virtual request may contain destinations of various different requests.

We now define the set of requests for the reoptimization problem to be the set of virtual requests and the set of nonassigned requests, and we enforce that each vehicle associated with an existing route starts its new route by picking up the virtual request.

3.1.3. Route Costs. The cost of a vehicle on a given day depends on the distance traveled and on the sleeping location of the driver on that day (recall that routes typically extend over several days). Let $L_r^k(t)$ be the distance traveled by vehicle k on route r on day t, and let $s_r^k(t)$ indicate whether the driver sleeps at his home location k^+ at the end of day t ($s^k(t) = 0$) or somewhere else ($s^k(t) = 1$). The amount that has to be paid to the driver is then equal to

$$\sum_t (\pi_k \max\{L^k(t), L^k_{\min}\} + \mu s^k(t)),$$

where π_k is the price per unit of distance traveled, $L^k_{\min}$ is the guaranteed minimum traveling distance that has to be paid, and μ is the compensation that has to be paid when a driver does not sleep at his home location. Note that a driver's pay includes the lease price of the vehicle.

These route costs cannot be used directly for the following reasons:

- The last part of a route may change in the future and therefore does not provide a reliable cost estimate.
- For a vehicle that is currently in use, the distance $L_r^k(0)$ will not exceed $L^k_{\min}$ during a significant part of the day, since initially only a subset of the requests is known. Similarly, for all vehicles, $L_r^k(t)$ will hardly ever exceed $L^k_{\min}$ for $t \geq 1$. Therefore, the route costs do not differentiate the various routes.

We use the following route costs. For a vehicle currently in use

$$c_r^k = \pi_k \sum_{0 \leq t \leq T_r} (\alpha \max\{L_r^k(t), L^k_{\min}\} + (1 - \alpha) L_r^k(t)) + \mu s_r^k(0),$$

and for all other vehicles

$$c_r^k = \pi_k \sum_{1 \leq t \leq T_r} (\alpha \max\{L_r^k(t), L^k_{\min}\} + (1 - \alpha) L_r^k(t)) + F,$$

for some $F \geq 0$ and $0 \leq \alpha \leq 1$, and where T_r denotes the last day that the vehicle is in use. Note that we do not take the sleeping locations of days $t \geq 1$ into account.

When we choose $\alpha = 1$, we obtain a cost structure that, during most of the current day, satisfies $c_r^k \approx (T_r + 1)\pi_k L^k_{\min} + \mu s_r^k(0)$ if the vehicle is currently in use, and $c_r^k \approx T_r \pi_k L^k_{\min} + F$ for all the other vehicles, which are almost independent of r. When we choose $0 < \alpha < 1$, we reflect that we want to make good short-term decisions, but still use some of the real cost structure in the model.

In practice, the values of $L^k_{\min}$ are much smaller than the average distance traveled per day per vehicle. If $\bar{L}$ is this average, then we have $\bar{L}/L^k_{\min} \approx 4/3$. The part of the objective function that deals with the costs associated with travel distance does not distinguish between a solution in

Table IX
Performance of Algorithms $A1$ and $A2$ for the Problem Class DAR50

		(CPU $\leq$ 600)			(CPU $\leq$ 1800)		
Algorithm	Z_{LP}	Z_{BEST}	#N	CPU	Z_{BEST}	#N	CPU
A_1	136,186.62	136,314.7	112.3	285.46	136,306.6	264.1	481.05
A_2	136,186.62	138,284.5	67.3	425.84	136,254.3	282.1	703.56

which four vehicles each travel a distance $L^k_{\min}$ and a solution in which three vehicles each travel a distance $\bar{L}$. The first solution, however, is unacceptable for Van Gend and Loos BV, because it would decrease the average income of the drivers, which would lead to an increase of the values π_k. Therefore, we have included the constant F in the cost for a vehicle currently in use. By taking the constant $F \gg \pi_k \alpha L^k_{\min}$, we discourage using a new vehicle. A new vehicle is introduced only when a new request becomes available that cannot be served by a vehicle currently in use.

3.1.4. The Reoptimization Problem. As indicated before, we can distinguish two phases in the planning process of each day. In the morning, the planners focus on the work that has to be done during that day. In the afternoon, the planners also consider the work that has to be done the next day and decide how many vehicles will be required to satisfy the (anticipated) transportation request for the next day.

We can easily accomplish that the known requests for the next day are being considered in the afternoon, by properly selecting the planning horizon H. However, since the objective function minimizes the number of vehicles and only a fraction of the requests that have to be served the next day are known, we must force a solution to use enough vehicles for the next day, so that new requests can be added when they become available.

We accomplish this by adding the following constraint to the set partitioning formulation

$$\sum_{k \in M} \sum_{r \in \Omega_k} Q_k x^k_r \geq \bar{Q},$$

where Ω_k is the set of feasible routes for vehicle k, Q_k is the capacity of vehicle k, x^k_r the variable that indicates whether route r for vehicle k is selected ($x^k_r = 1$) or not ($x^k_r = 0$), and $\bar{Q}$ an estimate of the total vehicle capacity that will be needed during the next day.

In the morning $\bar{Q}$ is set to a small value, say $\bar{Q} = 0$, effectively making the constraint redundant. At this time, the emphasis will be on short-term decisions.

3.2. Route Generation

The branch-and-price algorithm uses construction and improvement algorithms to generate additional routes. In this section, we discuss the modifications that had to be made to these algorithms to handle the specific characteristics of the direct transportation system: lunch breaks and night breaks.

A driver must have two breaks every day of a working period: a night break and a lunch break. Let Δ be the day length. So day t is the time interval $(t\Delta, (t+1)\Delta]$. A break on day t for vehicle k is characterized by a time window $[e, l] \subset (t\Delta, (t+1)\Delta]$ in which it must start, a duration σ, and a set V^k of locations at which the break can take place.

A night break has to be either at the vehicle's home location k^+ or at one of a set of common sleeping locations V_0. On Friday the night break has to be at the vehicle's home location. On Monday through Thursday the night break location depends on the predecessor and the successor in the route. Suppose we want a night break to take place between locations u and v, and suppose that vehicle k departs from u at time D_u. The night break takes place at a location $w \in V^k$ for which $D_u + t_{uw} \leq l$ and that minimizes the detour $d_{uw} + d_{wv}$.

A lunch break takes place immediately after arriving at a location on the vehicle's route, immediately before leaving a location on the vehicle's route, or at some location $j \in V_0 \cup \{k^+\}$ when traveling between two locations on the vehicle's route. The lunch location depends on the predecessor and the successor in the route. Suppose we want the lunch break to take place between location u and v, and suppose that vehicle k departs from u at time $D_u \leq l$. If $D_u \geq e$, lunch takes place at u. Otherwise, let $A_v = D_u + t_{uv}$. If $A_v \leq l$, then lunch takes place at v. Otherwise, lunch takes place at a location $w \in V_0 \cup \{k^+\}$ for which $D_u + t_{uw} \leq l$ and that minimizes the detour $d_{uw} + d_{wv}$. If $D_u + t_{uw} > l$ for all $w \in V_0 \cup \{k^+\}$, then lunch takes place in u.

The locations of the break stops are not determined a priori. They depend on the predecessor and the successor stops in the route. This implies that the insertion of a request into route r may lead to the relocation of some of these break stops. This obviously complicates computing the feasibility and cost associated with inserting a new request.

Suppose that route r for vehicle k ends at the end of day j. Inserting a new request into route r might require that the vehicle is also in use on day $j+1$. When we add a day at the end of a route, we must also introduce a lunch break and a night break for that day. This complicates the insertion algorithm. We therefore assume that, for some $T \geq 1$, all required breaks on days $0, 1, \ldots, T$, are already present in the route, regardless of whether the vehicle will be used on those days or not. We do not allow inserting a request after the last night break. T should be chosen large enough to ensure that this is not restrictive. In this way, for example, a route for a new vehicle that serves no requests consists of $2T$ stops: T lunch breaks and T night breaks.

3.3. Feasible Solutions

In a dynamic environment, a planning tool must be able to provide a good solution within a short amount of computation time. With an *LP* based branch-and-bound algorithm, we cannot guarantee that a good integral solution is found fast, if we rely only on the branching phase to produce these integral solutions. Therefore DRIVE frequently applies its primal heuristics. In fact, the primal heuristics are not only invoked when an *LP* has been solved to optimality, but each time a set of new routes is about to be generated, i.e., immediately before the pricing heuristics are activated.

Furthermore, DRIVE uses a heuristic construction algorithm in order to produce a starting solution very fast. This algorithm takes the solution of the previous reoptimization process as a starting point, and then sequentially assigns each new request j to the vehicle for which the marginal insertion cost is minimal. If it is infeasible to insert the new request in an existing route, then a new vehicle is introduced. After all new requests have been inserted into a

route, we try to improve the solution by applying three types of improvement algorithms. The first algorithm reinserts each request into its route. When, after request j has been inserted, new requests have been inserted into the same route, the current positions of the pickup and deliveries may no longer be optimal. Reinserting request j may therefore decrease the route cost. The other two algorithms take two routes r_1 and r_2, and try to find r_1' and r_2' such that $c_{r_1'}^{k_1} + c_{r_2'}^{k_2} < c_{r_1}^{k_1} + c_{r_2}^{k_2}$, either by moving requests from r_1 to r_2, or by exchanging requests between r_1 and r_2.

An advantage of using a set partitioning model in a dynamic environment is the fact that many of the routes that have been generated during the solution of the previous reoptimization problem are useful for the current reoptimization problem. Suppose that r is some route for vehicle k that has been generated in the previous reoptimization process, and suppose that r serves requests $j \in N_r$. We now create a new route for vehicle k as follows. Let r' be the route for vehicle k that only serves the virtual request i_k. Now sequentially insert all requests $j \in N_r$ that have not been permanently assigned to another vehicle, i.e., all requests $j \in N_r \cap N_0$, into route r'. The resulting routes provided a good set of initial routes for the branch-and-price algorithm.

4. CASE STUDY

We have tested DRIVE by simulating a dynamic planning environment with real-life data. These data contained all requests that had been served by Van Gend and Loos BV in a given period. These simulations were based on a stand-alone methodology. Solutions presented by DRIVE were not modified by a planner, but were considered as being executed as proposed.

During the development of DRIVE, we repeatedly compared the results of the simulations to the results of the planners at Van Gend and Loos BV over the same period. These comparisons were needed for various reasons. First, we had to make sure that the data were complete and that we were aware of all constraints that a route should satisfy. Second, we had to compare global solution characteristics, such as the number of vehicles used and the average distance traveled per vehicle per day, in order to assure that these were acceptable to Van Gend and Loos BV. Finally, we compared the total distance traveled and the total cost over the entire planning period, in order to show that DRIVE is capable of providing good solutions.

4.1. Organization of the Tests

The test data covered a period of 14 working days, starting with a Thursday and ending with a Tuesday. The actual evaluation period covered 10 working days, starting with a Monday and ending with a Friday. The two additional days at the beginning of the simulation were introduced in order to make sure that the vehicles are not empty on Monday morning, because that would not be realistic. The two additional days at the end of the simulation prevented DRIVE from deferring requests in order to obtain better results for the last days of the evaluation period. We present results only for the evaluation period.

Besides the usual request information, such as addresses of origins and destinations, time windows and load sizes, the test data also contained the times at which the requests became available, i.e., the times at which the requests were called in by the client. We used these data to simulate the process of clients calling in new requests.

The tests have been organized such that we have simulated the invocation of DRIVE once every hour, from 0600 to 1800, and once at midnight, for each day of the planning period. We assumed that the solution presented by DRIVE is then executed until DRIVE is invoked again. At 1500 we let DRIVE select the vehicles for the next day.

In practice, DRIVE must be able to produce a solution within a reasonable amount of time, such that a planner has enough time to evaluate the proposed solution. Therefore we have put an upper bound on CPU time of 5 minutes for each time we invoked DRIVE, with an exception for the run at 15:00, where the upper bound was 10 minutes.

4.2. Characteristics of the Reoptimization Problems

Table X shows the sizes of the reoptimization problems at times 0000, 0900, 1200, 1500, and 1800 for each day of the two planning weeks. For each reoptimization problem, we list the number of active requests $|N|$, the number of non-assigned requests $|N_1|$, and the number of vehicles $|M_0|$ currently in use. The set of active requests N has been defined by taking the planning horizon H equal to 19 hours. In this way we consider requests with an earliest pickup time of 0800 the next day, in the planning process starting at 1300 this day. When we take H much larger, DRIVE spends much CPU time in calculating assignments that will almost certainly have to be changed when new requests become available.

4.3. The Contribution of Column Generation

From our test results we observed that it is sometimes difficult to construct a feasible solution to a reoptimization problem, i.e., a solution in which all requests $j \in N_0$ are served. Note that we cannot always serve a new request by introducing a new vehicle. A new vehicle can start working only at the beginning of the next working day, so it cannot serve a new request that has to be picked up during the current day. The construction algorithm that we used to produce a starting solution could not always insert all new requests. At the beginning of the column generation algorithm, all requests that could not be inserted in any route during the creation of the starting solution get a very high dual value. Therefore the pricing heuristics will automatically try to construct routes that serve these requests.

4.4. Quality of Solutions

Tables XI and XII show the results of DRIVE compared to the results of the planners at Van Gend and Loos BV

Table X
Characteristics of the Reoptimization Problems for the Test Set

		Week 1					Week 2		
Day	Time	$\|N\|$	$\|N_1\|$	$\|M_0\|$	Day	Time	$\|N\|$	$\|N_1\|$	$\|M_0\|$
Mon	0000	328	197	92	Mon	0000	266	153	90
	0900	300	146	92		0900	261	111	90
	1200	285	134	92		1200	237	90	90
	1500	322	170	96		1500	227	104	87
	1800	329	175	96		1800	257	135	87
Tue	0000	349	204	96	Tue	0000	272	154	87
	0900	332	161	96		0900	249	118	87
	1200	325	162	96		1200	257	103	87
	1500	335	193	99		1500	271	137	90
	1800	333	178	101		1800	292	166	90
Wed	0000	354	208	101	Wed	0000	304	182	90
	0900	345	172	101		0900	272	118	90
	1200	306	140	101		1200	254	97	90
	1500	349	186	100		1500	256	122	86
	1800	341	174	100		1800	261	132	86
Thu	0000	354	195	100	Thu	0000	282	182	86
	0900	313	148	100		0900	264	118	86
	1200	292	125	100		1200	248	97	86
	1500	285	133	96		1500	261	144	92
	1800	285	140	96		1800	277	154	92
Fri	0000	306	166	96	Fri	0000	295	175	92
	0900	275	122	96		0900	278	113	92
	1200	264	99	96		1200	279	114	92
	1500	256	123	89		1500	281	147	95
	1800	244	125	90		1800	279	151	95

(VGL). For each day of the planning period we show the number of vehicles used, the total distance traveled, and the total cost of the solutions. The travel distances and the costs have been scaled, such that the total travel distance and the total cost per week equal 100.0 for the solution of Van Gend and Loos BV.

We observe that for the first planning week, DRIVE obtained an improvement of 4.7 percent of the total cost compared to the planners at Van Gend and Loos BV. For the second week, this improvement was 3.7 percent. The number of vehicles used by DRIVE is not substantially smaller than the number used by the planners at Van Gend and Loos BV. For the second week, it is even 7.7 percent higher. Apparently, DRIVE obtains the cost decrease by constructing better assignments of requests to vehicles.

The characteristics in Table X indicate that the first planning week is much busier than the second week. This is reflected in the difference between the solutions of DRIVE and the solutions of the planners. In a busy period, there are so many active requests that the planners must reduce the planning horizon in order to keep the problem manageable. At this time, DRIVE provides much better solutions, because it is able to look further into the future. In a quiet period, a planner has more time to make decisions, so he can better evaluate the effect of various assignments. In such a period, the cost decrease that DRIVE obtains is smaller but can still be significant.

5. CONCLUDING REMARKS

Although we tested DRIVE only in a simulated environment, the results indicate that DRIVE will provide a good basis for developing a decision support system at Van Gend and Loos BV. Clearly, the simulated environment in which we tested DRIVE was not entirely realistic. In practice, it

Table XI
Results for the First Planning Week

	No. of Vehicles		Travel Distance		Cost	
	VGL	DRIVE	VGL	DRIVE	VGL	DRIVE
Mon	92	92	20.1	18.2	20.4	18.6
Tue	97	96	19.7	19.9	20.1	20.5
Wed	100	101	20.1	20.1	20.4	20.7
Thu	100	100	20.8	18.8	21.0	19.2
Fri	97	96	19.3	16.1	18.1	16.3
Total	486	485	100.0	93.1	100.0	95.3

Table XII
Results for the Second Planning Week

	No. of Vehicles		Travel Distance		Cost	
	VGL	DRIVE	VGL	DRIVE	VGL	DRIVE
Mon	80	90	21.1	18.1	21.3	19.1
Tue	77	87	17.5	17.5	18.0	18.5
Wed	85	90	21.0	21.1	21.3	21.3
Thu	84	86	19.1	17.0	19.4	17.9
Fri	87	92	21.3	20.0	20.0	19.5
Total	413	445	100.0	93.7	100.0	96.3

sometimes happens that a vehicle breaks down, or that a load is not yet available at an origin at its indicated earliest pickup time. Such situations, which increase total cost, did not occur in our simulations. On the other hand, our tests have indicated that DRIVE, when implemented as a stand-alone system, is capable of providing solutions that are better than those provided by the planners at Van Gend and Loos BV. When DRIVE is embedded in a DSS, its solutions serve as a starting point for these planners. We may therefore expect that the cost decrease that we observed when DRIVE is used as a stand-alone system, will be even more significant when DRIVE is embedded in a DSS. Furthermore, the quality of the solutions produced by DRIVE may be increased by allowing more running time, switching to a faster computer, and using improved linear programming optimizers.

During the development of DRIVE, we identified several research topics related to dynamic pickup and delivery problems that deserve more attention in the future. First, we believe that there is not yet a good insight in objective functions for dynamic routing problems. Usually it is possible to define a cost structure that can be used to evaluation the cost of a route *after* it has been executed, but that cost structure does not always provide a good objective function for a reoptimization problem. A second research topic is the development of techniques for predicting requests that are not yet available. If we have some idea about the origin and load of future requests before they are actually called in, this might help in producing better plans.

ACKNOWLEDGMENT

We thank P. Maier, B. Muusers, and T. Steysiger of Van Gend and Loos BV and J. Hendriks of Logion BV for their help and support throughout the project. We also thank the Dutch Technology Foundation for their financial support.

APPENDIX

The appendix can be found at the *Operations Research* Home Page: http://opim.wharton.upenn.edu/~harker/opsresearch.html in the Online Collection.

REFERENCES

ANBIL, R., R. TANGA, AND E. L. JOHNSON. 1993. A Global Approach to Crew-Pairing Optimization. *IBM Systems J.* **31,** 71–78.

BARNHART, C., E. L. JOHNSON, G. L. NEMHAUSER, M. W. P. SAVELSBERGH, AND P. H. VANCE. 1998. Branch-and-Price: Column Generation for Solving Integer Programs. *Opns. Res.* **46,** 316–329.

CARPANETO, G. AND P. TOTH. 1980. Some New Branching and Bounding Criteria for the Asymmetric Travelling Salesman Problem. *Mgmt. Sci.* **26,** 736–743.

DESROCHERS, M., J. DESROSIERS, AND M. SOLOMON. 1992. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. *Opns. Res.* **40,** 342–354.

DESROSIERS, J., Y. DUMAS, M. M. SOLOMON, AND F. SOUMIS. 1996. Time Constrained Routing and Scheduling. In *Network Routing*, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser (eds.). North-Holland, 35–139.

DUMAS, Y., J. DESROSIERS, AND F. SOUMIS. 1991. The Pickup and Delivery Problem with Time Windows. *Euro. J. Opns. Res.* **54,** 7–22.

KINDERVATER, G. A. P. AND M. W. P. SAVELSBERGH. 1997. Vehicle Routing: Handling Edge Exchanges. E. H. L. Aarts and J. K. Lenstra (eds.). *Local Search in Combinatorial Optimization.* Wiley, Chichester, 337–360.

NEMHAUSER, G. L., M. W. P. SAVELSBERGH, AND G. C. SIGISMONDI. 1994. MINTO, a Mixed INTeger Optimizer. *O. R. Lett.* **15,** 47–58.

RYAN, D. M. AND B. A. FOSTER. 1981. An Integer Programming Approach to Scheduling. In *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*. North-Holland, Amsterdam, 269–280.

SAVELSBERGH, M. W. P. 1996. A Branch-and-Price Algorithm for the Generalized Assignment Problem. *Opns. Res.*

SAVELSBERGH, M. W. P. AND M. SOL. 1995. The General Pickup and Delivery Problem. *Transp. Sci.* **29,** 17–29.

SOL, M. 1994. Column Generation Techniques for Pickup and Delivery Problems. Ph.D. Thesis, Eindhoven University of Technology.

VANCE, P. H., C. BARNHART, E. L. JOHNSON, AND G. L. NEMHAUSER. 1994. Solving Binary Cutting Stock Problems by Column Generation and Branch and Bound. *Computational Optimiz. and Appl.* **3,** 111–130.