EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computing Science

Memorandum COSOR 94-22

A Branch-and-Price Algorithm for the Pickup and Delivery Problem with Time Windows

M. Sol
M.W.P. Savelsbergh

Eindhoven, July 1994
The Netherlands

A Branch-and-Price Algorithm for the Pickup and Delivery Problem with Time Windows

M. Sol
Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands

M.W.P. Savelsbergh
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205
U.S.A.

Abstract

In pickup and delivery problems vehicles have to transport loads from origins to destinations without transshipment at intermediate locations. In this paper, we describe an algorithm to solve such problems. The algorithm is based on a set partitioning formulation of the problem and uses new column generation and branching schemes.

1 Introduction

In the *Pickup and Delivery Problem* (PDP) 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) and the location where it is to be delivered (the destination). Each load has to be transported by one vehicle from its origin to its destination without any transshipment at other locations. A fleet of vehicles is available to operate the routes. The fleet may consist of various vehicle types. Each type specifies a vehicle capacity and a depot where the vehicles are stationed. In the *Pickup and Delivery Problem with Time Windows* (PDPTW) a transportation request may also specify time windows on both pickup and delivery. Furthermore there is, for each vehicle type, a time window specifying when the vehicles of that type are available. For a recent survey on pickup and delivery problems see Sol and Savelsbergh [10].

In this paper, we present a branch-and-price algorithm for the PDPTW that is based on a set partitioning formulation of the problem.

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 [1] and vehicle routing [4] problems. 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 [11] and generalized assignment [9] problems.

Branch-and-price algorithms [2] 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 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.

Dumas, Desrosiers and Soumis [5] have developed and implemented 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. We apply branching rules that focus on assignment rather than routing decisions, we develop various new algorithms for the pricing problem, and we embed these in a column management system. Furthermore, our algorithm has the advantage that it can be easily turned into an effective and efficient approximation algorithm.

Many of the ideas presented in this paper are not specific to pickup and delivery problems. In many optimization problems 'tasks' have to be assigned to 'resources'. These problems can all be formulated as a set partitioning problem, and the proposed branching strategy, the approximation algorithms for the pricing problem, as well as the column management system can be adapted easily to accommodate these problems.

The paper is organized as follows. Section 2 presents the set partitioning formulation of the PDPTW. Section 3 covers in detail the column generation scheme, the column management system, the algorithms for the pricing problem, and the branching strategy. Section 4 discusses the computational experiments. Finally, Section 5 presents some conclusions.

2 The set partitioning formulation

Let N be the set of transportation requests. For each transportation request $i \in N$, a load of size $q_i \in \mathbb{N}$ has to be transported from origin i^+ to destination i^-. Define $N^+ := \{i^+ \mid i \in N\}$ as the set of origins and $N^- := \{i^- \mid i \in N\}$ 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 vehicle types. Each vehicle of type $k \in M$ has a capacity $Q_k \in \mathbb{N}$, is available in the interval $[e_{k^+}, l_{k^+}]$, and is stationed at depot k^+. The number of available vehicles of type k is denoted by m_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 $V_k \setminus \{k^+\}$ is visited exactly once.*

5. *Each location in $V_k \setminus \{k^+\}$ 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 type 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:

$$\begin{array}{llll} \text{minimize} & \sum_{k \in M} \sum_{r \in \Omega_k} c_r^k x_r^k & & \\ \text{subject to} & \sum_{k \in M} \sum_{r \in \Omega_k} \delta_{ir}^k x_r^k = 1 & \text{for all } i \in N, & \text{(partitioning constraints)} \\ & \sum_{r \in \Omega_k} x_r^k \leq m_k & \text{for all } k \in M, & \text{(availability constraints)} \\ & x_r^k \in \{0,1\} & \text{for all } k \in M, r \in \Omega_k. & \end{array}$$

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 $c_{k^+j}^k = F + d_{k^+j}$ $(j \neq k^+)$.

3 A branch-and-price algorithm

We have developed a branch-and-price algorithm for the PDPTW based on the formulation presented in the previous section. 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 is used to solve LP.

A special branching strategy has been developed to solve P. Such a strategy is necessary to ensure that the pricing problem can be adjusted so that at no node of the branch-and-bound tree infeasible columns are generated.

3.1 Column generation

Suppose that for each vehicle type $k \in M$ 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:

$$\begin{array}{lll}
\text{minimize} & \sum\limits_{k\in M}\sum\limits_{r\in\Omega'_k} c_r^k x_r^k & \\
\text{subject to} & \sum\limits_{k\in M}\sum\limits_{r\in\Omega'_k} \delta_{ir}^k x_r^k = 1 & \text{for all } i \in N, \\
& \sum\limits_{r\in\Omega'_k} x_r^k \leq m_k & \text{for all } k \in M, \\
& x_r^k \geq 0 & \text{for all } k \in M,\ r \in \Omega'_k
\end{array}$$

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.,

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

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

$$\text{minimize}\{c_r^k - \sum_{i\in N} \delta_{ir}^k u_i - v_k \mid k \in M, r \in \Omega_k\}.$$

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 $z \geq 0$, 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*:

1. Find initial sets Ω'_k containing a feasible solution x.
2. Solve the restricted master problem LP'.
3. Solve the pricing problem. If $z \geq 0$ 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

$$\begin{array}{lll}
\text{minimize} & \sum_{k\in M}\sum_{r\in\Omega_k} c_r^k x_r^k + \sum_{i\in N} p y_i & \\
\text{subject to} & \sum_{k\in M}\sum_{r\in\Omega_k} \delta_{ir}^k x_r^k + y_i = 1 & \text{for all } i \in N \\
& \sum_{r\in\Omega_k} x_r^k \le m_k & \text{for all } k \in M \\
& x_r^k \ge 0 & \text{for all } k \in M, r \in \Omega_k \\
& y_i \ge 0 & \text{for all } i \in N
\end{array}$$

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

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

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

3.2 The pricing problem

The pricing problem decomposes into several independent problems, one for each vehicle type. Let S_k be the problem of minimizing $\{c_r^k - \sum_{i\in N} \delta_{ir}^k u_i - v_k \mid r \in \Omega_k\}$, i.e., the problem of finding a minimum cost route for a vehicle of type k, using a modified cost structure, that serves a subset of the transportation requests. To be more precise, introduce four types of variables: z_i $(i \in N)$ equal to 1 if transportation request i is served by the vehicle and 0 otherwise, x_{ij} $((i,j) \in V \times V)$ equal to 1 if the vehicle travels from location i to location j and 0 otherwise, D_i $(i \in V)$, specifying the departure time at location i, and y_i $(i \in V)$, specifying the load of the vehicle arriving at location i.

If the route r defined by vector x is a feasible pickup and delivery route for vehicle type k, its reduced cost can be expressed as follows:

$$d_r^k = \sum_{i \in V} \sum_{j \in V} c_{ij}^k x_{ij} - \sum_{i \in N} u_i z_i - v_k.$$

When we define $c'_{i^+j} := c^k_{i^+j} - u_i$, $c'_{k^+j} := c^k_{k^+j} - v_k$ and $c'_{ij} := c^k_{ij}$ if $i \notin N^+ \cup \{k^+\}$, then $d_r^k = \sum_{i \in V} \sum_{j \in V} c'_{ij} x_{ij}$. Define $q_{k^+} = 0$. The pricing problem S_k is to

minimize $\sum_{i \in V} \sum_{j \in V} c'_{ij} x_{ij}$
subject to

$\sum_{j \in V} x_{i^+j} = \sum_{j \in V} x_{ji^+} = z_i$	for all $i \in N$,	(1)
$\sum_{j \in V} x_{i^-j} = \sum_{j \in V} x_{ji^-} = z_i$	for all $i \in N$,	(2)
$\sum_{j \in V} x_{k^+j} = 1$,		(3)
$\sum_{i \in V} x_{ik^+} = 1$,		(4)
$D_{i^+} \leq D_{i^-}$	for all $i \in N$,	(5)
$x_{ij} = 1 \Rightarrow D_i + t_{ij} \leq D_j$	for all $i, j \in V$,	(6)
$y_{k^+} = 0$,		(7)
$x_{ij} = 1 \Rightarrow y_j = y_i + q_i$	for all $i, j \in V$,	(8)
$x_{ij} \in \{0, 1\}$	for all $i, j \in V$,	(9)
$z_i \in \{0, 1\}$	for all $i \in N$,	(10)
$e_i \leq D_i \leq l_i$	for all $i \in V$,	(11)
$0 \leq y_i \leq Q_k$	for all $i \in V$.	(12)

By constraints (1) and (2) a vehicle visits either both the origin and the destination of a request or it visits neither of them. Constraints (3) and (4) make sure that the vehicle starts and ends at its depot. Constraints (5), (6), and (11) together form the time and precedence constraints. Constraints (7), (8), and (12) together form the capacity constraints.

Problem S_k can be viewed as a shortest path problem with precedence constraints, capacity constraints and time windows, on the perturbed distance matrix c'. Dumas, Desrosiers and Soumis [5] propose a dynamic programming algorithm, which uses labeling techniques to handle the precedence, capacity and time constraints, to solve this shortest path problem. The state space used in this algorithm consists of labels $(l, R(S), T, D)$, where l is the last location visited on the path corresponding to this label, S is the set of requests that are picked up on the path, $R(S) \subseteq S$ is the set of requests that has not been delivered yet, T is the departure time at l and D is the reduced cost of the path. The size of this state space is exponential, due to the presence of $R(S)$ in the labels. Furthermore, by storing $R(S)$ instead of S in each label, the constraint $z_i \in \{0, 1\}$ is relaxed to $z_i \in \mathbb{N}$, allowing the pricing algorithm to produce a route that serves requests more than once. Although such routes never appear in an integral solution of P, they might decrease Z_{LP}, the optimal value of LP, thus inducing a weaker lower bound than would be found with the restriction $z_i \in \{0, 1\}$.

For instances with many transportation requests and time and capacity constraints that are not very tight, solving S_k to optimality becomes computationally prohibitive.

3.3 Column management

Observe that any column $r \in \Omega_k$ with negative reduced cost is a candidate to enter the basis and may therefore be added to the restricted master problem. Consequently, it is not necessary to solve the pricing problem to optimality as long as a column with negative reduced cost can be found. Furthermore, if more than one column with negative reduced cost is found, these columns may be added simultaneously to the restricted master problem.

Based on these observations, we propose a column management system that uses approximation as well as optimization algorithms for the pricing problem. The approximation algorithms try to generate many routes with negative reduced cost very fast. These routes are then 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 $D_{\max} \geq 0$. A positive treshold value can be useful, because the reduced costs change after every reoptimization, possibly to a negative value. If the pricing problem is not solved to optimality, LP cannot be solved to optimality and so P cannot be solved to optimality. Therefore, as soon as the approximation algorithms fail to find columns with negative reduced cost, an optimization algorithm is used to solve the pricing problem to either prove optimality or to find new columns. The column generation scheme we propose now looks as follows:

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

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

3. Solve the restricted master problem LP'.

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.

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.

6. Solve the pricing problem to optimality. If $z \geq 0$, 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.

3.4 Solving the linear program

As pointed out in the previous section, it is not necessary to solve the pricing problem by an optimization algorithm as long as columns with negative reduced cost can be found by an approximation algorithm. Because the optimization algorithm, which is based on dynamic programming, is very time consuming, we discuss various strategies to solve the linear program efficiently. Note that to prove optimality of an LP solution we need to solve the pricing problem optimally.

Dumas, Desrosiers and Soumis [5] propose to speed up the dynamic programming algorithm in earlier iterations of the column generation process by working on a reduced network. 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.

Observe 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 also guarantees that in the end the pricing problem is solved to optimality.

A slightly different strategy is to use polynomial approximation algorithms for the pricing problem as long as they can generate columns with negative reduced costs and only use a dynamic programming algorithm when the approximation algorithms fail to generate a column with negative reduced cost. The approximation algorithms we have used are based on construction and improvement algorithms for the single-vehicle PDPTW. In their description, we use $I_r^k(j)$ to denote the minimal cost of inserting transportation request j into route r. 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 $\mathcal{O}(n^2)$ 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 $\mathcal{O}(n^2)$ 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 $r \setminus \{i\}$ 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:

1. Let $d^k_{(r\setminus\{i_0\})\cup\{j_0\}} = \min\{d^k_{(r\setminus\{i\})\cup\{j\}} \mid i, j \in N, \delta^k_{ir} = 1, \delta^k_{jr} = 0\}$.
2. If $d^k_{(r\setminus\{i_0\})\cup\{j_0\}} < d_r$, then set $r = (r \setminus \{i_0\}) \cup \{j_0\}$ and go to Step 1.
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:

1. $d_1 = \infty$, $F = \emptyset$.
2. Let $d^k_{(r\setminus\{i_0\})\cup\{j_0\}} = \min\{d^k_{(r\setminus\{i\})\cup\{j\}} \mid i,j \in N, \delta^k_{ir} = 1, \delta^k_{jr} = 0, j \notin F\}$.
3. If $d^k_{(r\setminus\{i_0\})\cup\{j_0\}} = \infty$, then go to Step 6, otherwise set $r = (r \setminus \{i_0\}) \cup \{j_0\}$ and $F = F \cup \{i_0\}$.
4. If $d_r < d_1$, then set $d_1 = d_r$ and $r_1 = r_0$.
5. Go to Step 2.
6. If $d_1 < 0$, then add r_1 to the column pool.

3.5 Branching schemes

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.

3.5.1 Branching on routing decisions

Dumas, Desrosiers and Soumis [5] 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 [3]. Let $x^k_r > 0$ be a fractional variable in the current LP solution. Suppose that the corresponding route r serves n requests $\{i_1, i_2, ..., i_n\} \subset N$ 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 $j \in \{0, 1, ..., n\}$ 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.

3.5.2 Branching on assignment decisions

We present two branching schemes that focus on assignment decisions. They are both special cases of a branching scheme proposed by Ryan and Foster [8] for pure set partitioning problems, which is based on the following proposition:

Proposition 1 *Let A be a $0-1$ matrix of size $m \times n$ and let x be a fractional basic solution to $Ax = 1$, $x \geq 0$. Then there exist two rows p and q such that $0 < \sum_{j=1}^{n} A_{pj}A_{qj}x_j < 1$.*

The set of integral solutions to $Ax = 1$ can now be divided into two subsets, characterized by $x_j = 0$ if $A_{pj} + A_{qj} = 1$, and $x_j = 0$ if $A_{pj} + A_{qj} \neq 1$, respectively. In the first subset p and q must be covered by the same column. In the second subset p and q must be covered by two distinct columns.

In our set partitioning model for the pickup and delivery problem, we have two sets of rows, corresponding to the partitioning constraints for the clients and the availability constraints for the vehicles. When both p and q are partitioning rows, the Ryan and Foster branching scheme implies the creation of a subset in which transportation requests p and q must be served by the same vehicle, and another subset in which both requests must be served by two different vehicles. When p is a set partitioning row and q is the availability constraint of vehicle type k, the first subset consists of solutions where transportation request p is served by vehicle type k, while in the second subset transportation request p may not be served by vehicle type k. One can easily see that it is impossible that both p and q correspond to availability constraints.

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 type k in the current LP solution, and define for all $i, j \in N$ the *combination value* $y_{ij} = \sum_{k\in M}\sum_{r\in\Omega_k} \delta_{ir}^k \delta_{jr}^k x_r^k$, indicating what fraction of the routes serving i also serves j. We have the following two lemmas.

Lemma 1 *Suppose $m_k = 1$ for all $k \in M$. 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 and so they are identical. Since all routes in Ω_k are distinct, this implies that x is an integral solution. $\square$

Note that we can always enforce the restriction $m_k = 1$ for all $k \in M$, by duplicating each vehicle type k m_k times. This is however a pseudopolynomial transformation.

Lemma 2 *Let x be an optimal solution of LP, and let $y_{ij} = \sum_{k\in M}\sum_{r\in\Omega_k} \delta_{ir}^k \delta_{jr}^k x_r^k$ $(i, j \in N)$. If y is integral, then x is a convex combination of integral solutions of LP.*

Proof Suppose that y is integral and let $y_{ij} = 1$. Then $\sum_{k\in M}\sum_{r\in\Omega_k} \delta_{ir}^k \delta_{jr}^k x_r^k = 1 = \sum_{k\in M}\sum_{r\in\Omega_k} \delta_{ir}^k x_r^k = \sum_{k\in M}\sum_{r\in\Omega_k} \delta_{jr}^k x_r^k$. This implies that $\delta_{ir}^k = \delta_{jr}^k$ for all $k \in M$

and $r \in \Omega_k$ with $x_r^k > 0$. Let $\overline{\Omega}_k = \{r \in \Omega_k | x_r^k > 0\}$, $\overline{\Omega} = \bigcup_{k \in M} \overline{\Omega}_k$, and $Q = \{\xi \mid \sum_{k \in M} \sum_{r \in \overline{\Omega}_k} \delta_{ir}^k \xi_r^k = 1 \;\; (i \in N),\; \sum_{r \in \overline{\Omega}_k} \xi_r^k \leq m_k \;\; (k \in M),\; \xi_r^k \geq 0\}$. Observe, that the restriction of x to $\overline{\Omega}$, say $\overline{x}$, is an element of Q. Furthermore, the constraint matrix A defining Q is totally unimodular, since, after deletion of duplicate rows, each column of A contains exactly two 1 entries; one in the partitioning rows and one in the availability rows. Consequently, Q is an integral polyhedron and $\overline{x}$, and thus x, are convex combinations of integral solutions of LP. □

Corollary 1 *Let x be an optimal extremal solution of LP. Let $y_{ij} = \sum_{k \in M} \sum_{r \in \Omega_k} \delta_{ir}^k \delta_{jr}^k x_r^k$ $(i, j \in N)$. Then x is integral if and only if y is integral.*

Note that in Lemma 1, Lemma 2 and Corrolary 1 we can replace LP by the restricted master problem LP'. This ensures that the branching schemes that are based on these results are also valid when LP is not solved to optimality.

The two branching schemes we propose can be summarized as follows:

Scheme 1: When x is fractional, there are a request $\hat{\imath} \in N$ and a vehicle type $\hat{k} \in M$ with $0 < z_{\hat{\imath}}^{\hat{k}} < 1$. Create two subsets characterized by $z_{\hat{\imath}}^{\hat{k}} = 0$ and $z_{\hat{\imath}}^{\hat{k}} = 1$ respectively.

Scheme 2: When x is fractional, there are two requests $\hat{\imath}, \hat{\jmath} \in N$ such that $0 < y_{\hat{\imath}\hat{\jmath}} < 1$. Create two subsets characterized by $y_{\hat{\imath}\hat{\jmath}} = 0$ and $y_{\hat{\imath}\hat{\jmath}} = 1$ respectively.

When the number of vehicles of a single type is large, i.e. m_k is large, the pseudopolynomial transformation that is needed in order to use scheme 1, will drastically increase problem size. On the other hand, we will show that scheme 1 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 trivial to see that the restriction $z_{\hat{\imath}}^{\hat{k}} = 1$, i.e. requiring that request $\hat{\imath}$ is served by vehicle $\hat{k}$, does not complicate the pricing problem either. In fact, any algorithm for the pricing problem $S_{\hat{k}}$ can be used, as is stated by the following theorem.

Theorem 1 *Let $S_{\hat{k}}(\hat{\imath})$ be the problem that arises from $S_{\hat{k}}$ by adding the constraint $z_{\hat{\imath}}^{\hat{k}} = 1$, and let A be any algorithm that solves $S_{\hat{k}}$. Then A also solves $S_{\hat{k}}(\hat{\imath})$.*

Proof The idea behind the proof is to modify the dual variables such that they still form an optimal dual solution, but all routes $r \in \Omega_{\hat{k}}$ with $\delta_{\hat{\imath}r}^{\hat{k}} = 0$ will have $d_r^{\hat{k}} \geq 0$.

Note that request $\hat{\imath}$ can be forced to be in vehicle $\hat{k}$ by adding the following branching constraint to the master problem:

$$\sum_{r \in \Omega_{\hat{k}}} (1 - \delta_{\hat{\imath}r}^{\hat{k}}) x_r^{\hat{k}} + \sum_{k \in M \setminus \{\hat{k}\}} \sum_{r \in \Omega_k} \delta_{\hat{\imath}r}^{k} x_{\hat{\imath}}^{k} \leq 0$$

Let w be the dual variable associated with this branching constraint. When the restricted master problem has been solved, the reduced costs of the columns in the restricted master problem satisfy

$$d_r^k = c_r^k - \sum_{i \in N} \delta_{ir}^k u_i - v_k - \delta_{\hat{\imath}r}^k w \geq 0 \qquad \text{if } r \in \Omega_k', k \in M \setminus \{\hat{k}\}$$

and

$$d_r^{\hat{k}} = c_r^{\hat{k}} - \sum_{i \in N} \delta_{ir}^{\hat{k}} u_i - v_{\hat{k}} - (1 - \delta_{\hat{i}r}^{\hat{k}}) w \geq 0 \qquad \text{if } r \in \Omega'_{\hat{k}}$$

Let $\eta > 0$ and consider the following alternative dual solution: $\overline{u} := u + \eta e_{\hat{i}}, \overline{v} := v - \eta e_{\hat{k}}$, and $\overline{w} := w - \eta$, where e_j denotes the jth unit vector. This alternative dual solution is feasible, since the modified reduced costs $\overline{d_r^k}$ of the columns in the current restricted master problem satisfy

$$\overline{d_r^k} = c_r^k - \sum_{i \in N} \delta_{ir}^k \overline{u}_i - \overline{v}_k - \delta_{\hat{i}r}^k \overline{w} = c_r^k - \sum_{i \in N} \delta_{ir}^k u_i - v_k - \delta_{\hat{i}r}^k w = d_r^k \geq 0$$

if $r \in \Omega'_k, k \in M \setminus \{\hat{k}\}$,

$$\overline{d_r^{\hat{k}}} = c_r^{\hat{k}} - \sum_{i \in N} \delta_{ir}^{\hat{k}} \overline{u}_i - \overline{v}_{\hat{k}} - (1 - \delta_{\hat{i}r}^{\hat{k}}) \overline{w} = c_r^{\hat{k}} - \sum_{i \in N} \delta_{ir}^{\hat{k}} u_i - v_{\hat{k}} - (1 - \delta_{\hat{i}r}^{\hat{k}}) w = d_r^{\hat{k}} \geq 0$$

if $r \in \Omega'_{\hat{k}}$ and $\delta_{\hat{i}r}^{\hat{k}} = 1$, and

$$\overline{d_r^{\hat{k}}} = c_r^{\hat{k}} - \sum_{i \in N} \delta_{ir}^{\hat{k}} \overline{u}_i - \overline{v}_{\hat{k}} - (1 - \delta_{\hat{i}r}^{\hat{k}}) \overline{w} = c_r^{\hat{k}} - \sum_{i \in N} \delta_{ir}^{\hat{k}} u_i - v_{\hat{k}} - (1 - \delta_{\hat{i}r}^{\hat{k}}) w + 2\eta$$

$$= d_r^{\hat{k}} + 2\eta \geq 0$$

if $r \in \Omega'_{\hat{k}}$ and $\delta_{\hat{i}r}^{\hat{k}} = 0$. Clearly, the alternative dual solution is also optimal, because $\sum_i \overline{u}_i + \sum_k \overline{v}_k = \sum_i u_i + \sum_k v_k$. It should be obvious that by choosing η large enough, we can ensure that $\overline{d_r^{\hat{k}}} \geq 0$ for all $r \in \Omega_{\hat{k}}$ with $\delta_{\hat{i}r}^{\hat{k}} = 0$. □

Note that this result for branching scheme 1 holds for general set partitioning problems with availability constraints. For branching scheme 2 there is no such a general result known. For the pickup and delivery problem with time windows, however, it is not difficult to adjust the algorithms for the pricing problem, such that they only produce solutions that satisfy all combination restrictions.

3.6 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 $z_i^k = \sum_{r \in \Omega_k} \delta_{ir}^k x_r^k$. The following algorithm now tries to construct a feasible solution:

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.

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

Table 1: Problem classes

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

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

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 [6].

4 Computational experiments

The ultimate goal of our research is the development of a high quality approximation algorithm for the PDPTW that can solve fairly large 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.

To be able to determine the impact of the various choices that are inherently available in our branch-and-price algorithm, e.g., whether to use approximation or optimization algorithms for the pricing problem, we have implemented several versions of our algorithm. We start our description of these algorithms with a discussion of the implementation issues that are common to all versions.

The cheapest insertion algorithm that we developed for the approximation algorithms for the pricing problem is also used to construct a starting solution and a first set of columns for the set partitioning matrix.

The bound on the number of columns that the dynamic programming algorithm can generate for each vehicle in a single execution was set to 200. We experimented

Problem	Z_{opt}	Z_{LP}	gap	Problem	Z_{opt}	Z_{LP}	gap
A30.1	104291	104263.0	0.65	B30.1	124655	124655.0	0
A30.2	114451	114451.0	0	B30.2	94341	94332.0	0.21
A30.3	135415	135415.0	0	B30.3	124882	124872.0	0.21
A30.4	115148	115148.0	0	B30.4	104431	104431.0	0
A30.5	114535	114535.0	0	B30.5	94434	94403.0	0.70
A30.6	114967	114939.8	0.55	B30.6	104264	104264.0	0
A30.7	104247	104247.0	0	B30.7	104263	104262.0	0.02
A30.8	115154	115146.0	0.16	B30.8	135332	135332.0	0
A30.9	114883	114883.0	0	B30.9	145466	145466.0	0
A30.10	115208	115208.0	0	B30.10	114685	114685.0	0
C30.1	84664	84604.0	1.29	D30.1	94460	94440.5	0.44
C30.2	84674	84674.0	0	D30.2	73855	73832.0	0.60
C30.3	84011	83961.7	1.23	D30.3	84737	84737.0	0
C30.4	74205	74168.7	0.86	D30.4	84765	84765.0	0
C30.5	84426	84391.2	0.79	D30.5	84593	84498.5	2.06
C30.6	94547	94500.5	1.02	D30.6	84257	84247.5	0.22
C30.7	84270	84187.4	1.93	D30.7	84302	84278.5	0.55
C30.8	94778	94767.0	0.23	D30.8	74293	74097.0	4.57
C30.9	84246	84231.2	0.35	D30.9	84110	84110.0	0
C30.10	84230	84213.2	0.40	D30.10	74499	74402.5	2.14

Table 2: Optimal solutions and integrality gaps

with upper bounds of 50, 200 and 1000 columns per vehicle, but the computation times hardly varied. Because the upper bound of 200 columns per vehicle produced slightly better results, this value was choosen. The threshold value used when the column pool is cleaned up has been set to 10.

All versions of the algorithm use branching scheme 1, i.e., they branch on assignment decisions of transportation requests to vehicles. The branching pair $(\hat{k}, \hat{\imath})$ is chosen such that $z_{\hat{\imath}}^{\hat{k}} = \max\{z_i^k \mid z_i^k < 1, i \in N, k \in M\}$. The search tree is explored according to a best bound search. We did not implement scheme 2, because the search trees turned out to be fairly small and the effect of a different branching scheme would be minimal.

The first two versions of our algorithm, O_1 and O_2, are optimization algorithms. Both of them use the column generation scheme of Section 3.2, but O_1 only uses the dynamic programming algorithm for column generation, whereas O_2 solves the pricing problem approximately as long as this gives profitable routes.

The second two versions of our algorithm, A_1 and A_2, are approximation algorithms. Both of them use the column generation scheme of Section 3.2, but A_1 never uses the dynamic programming algorithm and therefore never solves the pricing problem to optimality, whereas A_2 solves the pricing problem optimally in the root when the approximation algorithms fail to produce profitable columns. Note that in the root A_2 is equivalent to O_2 and solves the LP to optimality. Therefore it provides a valid lower bound on the optimal solution value. In none of the other nodes A_1 or A_2 guarantee that the LPs are solved to optimality. This implies that for these nodes the lower bounds are not always valid, possibly resulting in the deletion of the node containing the optimal solution.

For comparison purposes, we have also included the more traditional approximation algorithm, A_3, that circumvents the complications introduced by branching by only generating columns at the root node and then performing standard branching based on variable dichotomy with the given set of columns.

All versions have been implemented using MINTO, a Mixed INTeger Optimizer [7]. 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 1.6/CPLEX 2.1 and have been run on an IBM/RS6000 model 550.

4.1 Test problems

As the size of the set of all feasible solutions to an instance of the pickup and delivery problem with time windows 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, and this size may have a strong relationship with the ability of our algorithms to solve instances, 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 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 $F = 10000$.

Table 1 lists the problem classes that we have used in our experiments in order of anticipated difficulty. We have randomly generated 10 instances in each problem class.

4.2 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 $\sum_{k\in M}\sum_{r\in\Omega_k} x_r^k \geq m$ (see Section 3.1). 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 2 shows the linear programming bound at the root (Z_{LP}), the value of the optimal solution (Z_{OPT}), and the integrality gap with respect to distance traveled only: $(Z_{OPT} - Z_{LP})/(Z_{OPT} - mF)$. 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%.

	O_1			O_2			
Problem	CPU	generated	added	CPU	generated	added	
A30.1	197	9342	216	145	1613	266	0.74
A30.2	32	5747	204	26	885	204	0.81
A30.3	17	2006	153	17	962	190	1.00
A30.4	54	4234	180	31	933	221	0.57
A30.5	51	5392	190	30	791	181	0.59
A30.6	28	3490	182	22	944	202	0.79
A30.7	51	5496	193	33	897	190	0.65
A30.8	18	3282	227	18	862	224	1.00
A30.9	23	3640	198	15	1033	215	0.65
A30.10	28	4215	215	21	922	193	0.75
B30.1	34	3474	162	21	571	165	0.62
B30.2	89	7097	218	53	1103	235	0.60
B30.3	30	3447	161	25	690	170	0.83
B30.4	61	7600	188	32	936	214	0.54
B30.5	162	9640	217	90	938	214	0.56
B30.6	57	5460	191	33	930	226	0.58
B30.7	60	5383	217	40	975	229	0.67
B30.8	15	2015	183	12	800	171	0.80
B30.9	28	4502	165	25	807	213	0.89
B30.10	33	5770	165	36	934	193	1.09
C30.1	145	10180	248	111	1328	290	0.77
C30.2	418	14781	272	240	1248	283	0.57
C30.3	719	15332	281	316	1421	267	0.44
C30.4	914	18152	340	469	1550	302	0.51
C30.5	612	13270	301	276	1627	286	0.45
C30.6	152	10812	241	99	1361	267	0.65
C30.7	837	11234	305	499	1418	298	0.60
C30.8	295	10453	297	210	1352	284	0.71
C30.9	487	16374	319	305	1647	281	0.63
C30.10	825	14993	301	702	1852	323	0.85
D30.1	498	12187	240	255	1438	257	0.51
D30.2	1903	14880	287	1159	1265	296	0.61
D30.3	335	10523	272	206	1198	244	0.61
D30.4	503	13332	234	191	1399	280	0.38
D30.5	269	9740	287	151	1143	298	0.56
D30.6	341	10398	275	209	1269	274	0.61
D30.7	533	12912	315	226	1532	289	0.42
D30.8	1110	16782	321	651	1359	268	0.59
D30.9	1094	14707	332	614	1864	306	0.56
D30.10	399	15041	293	165	1502	248	0.41

Table 3: Performance of optimization algorithms for Z_{LP}

Class	A30		B30		C30		D30	
Nr	CPU	nodes	CPU	nodes	CPU	nodes	CPU	nodes
1	1497	55	21	1	1081	39	937	9
2	26	1	121	5	240	1	13209	37
3	17	1	74	5	4752	57	206	1
4	31	1	32	1	9128	87	191	1
5	30	1	662	23	5158	77	12488	571
6	265	45	33	1	1436	75	955	19
7	33	1	59	3	18391	201	840	13
8	40	5	12	1	498	9	253549	1845
9	15	1	25	1	1490	21	614	1
10	21	1	36	1	7143	43	6983	109

Table 4: Performance of optimization algorithm O_2

4.3 Performance of the optimization algorithms

To analyze the effect of using approximation algorithms in the column generation scheme, we have evaluated the root, i.e., solved the linear program, by both algorithm O_1 and O_2 for all instances in the problem classes A30, B30, C30 and D30. Table 3 shows the CPU time, the number of columns generated, and the number of columns added by O_1 and O_2. 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% 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 poolsize for O_1 than for O_2, the larger poolsize does not cause the differences in computation times. The differences in computation times can be fully attributed to the fact that O_1 only uses 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 4 shows the total CPU time and the number of nodes evaluated in the search tree.

4.4 Performance of the approximation algorithms

Table 5 shows the CPU time, the number of evaluated nodes, and the relative error $(Z_{BEST} - Z_{OPT})/(Z_{OPT} - mF)$ for the approximation algorithms A_1, A_2, and A_3. When a problem is solved in the root, A_2 and A_3 are equivalent to O_2. This event is indicated by an asterisk (*) in the last column.

	A_1			A_2			A_3			
Problem	CPU	nodes	error	CPU	nodes	error	CPU	nodes	error	
A30.1	158	14	0	187	27	0	149	11	0.09	
A30.2	23	4	1.37	26	1	0	26	1	0	*
A30.3	7	1	1.70	17	1	0	17	1	0	*
A30.4	13	1	1.07	31	1	0	31	1	0	*
A30.5	16	1	0.04	30	1	0	30	1	0	*
A30.6	58	29	0	66	41	0	23	17	0.46	
A30.7	11	1	4.87	33	1	0	33	1	0	*
A30.8	15	2	0.35	19	3	0	19	5	0.10	
A30.9	13	1	1.00	15	1	0	15	1	0	*
A30.10	17	3	0.31	21	1	0	21	1	0	*
B30.1	10	1	2.77	21	1	0	21	1	0	*
B30.2	16	1	0.25	61	5	0	54	5	0	
B30.3	10	1	2.03	30	5	0.20	26	5	0.20	
B30.4	13	1	0.14	32	1	0	32	1	0	*
B30.5	50	11	0.07	108	13	0	93	17	0	
B30.6	15	1	2.18	33	1	0	33	1	0	*
B30.7	20	1	2.65	43	3	0	41	3	0	
B30.8	8	1	0.75	12	1	0	12	1	0	*
B30.9	12	3	1.50	25	1	0	25	1	0	*
B30.10	9	1	1.71	36	1	0	36	1	0	*
C30.1	164	41	0.90	250	69	0	114	9	0.75	
C30.2	33	2	0	240	1	0	240	1	0	*
C30.3	27	1	2.14	380	25	0	321	13	0	
C30.4	222	53	1.07	525	29	0	482	21	0.19	
C30.5	64	9	0.34	424	67	0.18	283	25	0.54	
C30.6	75	21	0	209	71	0	102	21	0	
C30.7	327	102	0	804	127	0	505	15	0	
C30.8	63	11	0.17	215	4	0	216	31	0.50	
C30.9	48	6	0.31	387	19	0.02	309	9	0.12	
C30.10	103	15	0	756	21	0	722	7	0	
D30.1	36	3	0.85	267	9	0	258	3	0	
D30.2	68	12	0.60	1214	21	0	1190	8	0	
D30.3	22	2	2.05	206	1	0	206	1	0	*
D30.4	23	1	3.44	191	1	0	191	1	0	*
D30.5	1540	596	0	1281	465	0	166	107	0.09	
D30.6	23	1	2.37	300	57	0.33	215	23	0.68	
D30.7	26	1	0.93	243	11	0	230	3	0	
D30.8	984	277	0	725	31	0	658	23	0.63	
D30.9	38	1	2.85	614	1	0	614	1	0	*
D30.10	120	14	0	406	101	0	167	11	0.60	

Table 5: Performance of approximation algorithms

4.4.1 Quality

Algorithm A_2 clearly outperforms the others with respect to quality of the solutions. It solves 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 only solved to optimality in the root node. For the four problems that were not solved to optimality, the relative

Class	$\lvert N\rvert$	$\lvert M\rvert$	$q^{\min}$	$q^{\max}$	Q	W
A50	50	25	5	15	15	60
B50	50	25	5	20	20	60
DAR30	30	15	1	1	5	60

Table 6: Larger and less restricted problem classes

error was at most 0.33 %.

By comparing the results of A_2 and A_3, we conclude that it pays off 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.

4.4.2 Speed

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 1.07%. 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 the other algorithms cannot be used because of the computation times of the dynamic programming algorithm.

4.4.3 Difficult instances

In a final experiment, we have generated two sets of larger instances and one set of less restricted instances and tested algorithms A_1 and A_2 on these sets. The characteristics of these problem classes are shown in Table 6. The set DAR30 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. Let Z_{BEST} denote the best solution found by the algorithm. The relative error is then approximated by $(Z_{BEST} - \lceil Z_{LP} \rceil)/(\lceil Z_{LP} \rceil - mF)$, which is an upper bound on the real relative error. We introduced an upper bound of 2500 on the number of nodes that may be evaluated in the search tree. When this bound causes the algorithm to stop, Z_{BEST} is the best solution found so far.

The results for classes A50 and B50 are shown in Table 7 and the results for the class DAR30 are shown in Table 8. They indicate that the algorithms are indeed capable of providing a high quality solution in an acceptable amount of computation time.

5 Concluding remarks

We have presented a branch-and-price algorithm based on a set partitioning formulation that is capable of solving moderately sized pickup and delivery problems to optimality. Furthermore, the algorithm can easily be turned into an efficient and effective approximation algorithm to solve larger instances.

		A_1				A_2			
Problem	Z_{LP}	CPU	nodes	Z_{BEST}	error	CPU	nodes	Z_{BEST}	error
A50.1	177123.0	76	4	177136	0.18	271	1	177123	0
A50.2	177628.0	176	16	177659	0.41	642	39	177645	0.22
A50.3	167762.0	65	1	167888	1.62	129	1	167762	0
A50.4	147703.5	60	1	148040	4.36	353	5	147708	0.05
A50.5	208126.0	27	1	208160	0.42	86	1	208126	0
A50.6	156855.4	194	14	156887	0.45	1150	127	156867	0.16
A50.7	177035.0	249	20	177061	0.37	437	9	177061	0.37
A50.8	156875.0	98	3	156989	1.66	627	1	156875	0
A50.9	146728.0	222	9	146742	0.21	836	7	146736	0.12
A50.10	158170.0	71	1	158516	4.24	1424	205	158219	0.60
B50.1	147058.0	104	3	147142	1.19	484	1	147058	0
B50.2	177201.7	59	1	177211	0.12	395	37	177211	0.12
B50.3	137144.8	9566	2500 *	137567	5.91	9671	2500 *	137330	2.59
B50.4	146634.0	64	1	146872	3.59	425	1	146634	0
B50.5	168343.7	4761	990	168452	1.29	10468	2500 *	168398	0.65
B50.6	156908.9	904	118	156996	1.26	4687	731	156941	0.46
B50.7	166413.3	391	33	166426	0.19	477	31	166422	0.12
B50.8	187756.0	99	6	187896	1.81	256	21	187767	0.14
B50.9	156927.5	241	15	156957	0.42	803	81	156951	0.33
B50.10	157074.0	12055	2500 *	157266	2.71	7654	2500 *	157250	2.49

Table 7: Performance of algorithms $A1$ and $A2$ for the problem classes A50 and B50

A more traditional use of set partitioning formulations for routing problems has been to generate a large set of columns heuristically and then solve the resulting set partitioning problem over this set of columns. Our computational results, specifically those relating to algorithm A_3 (see Section 5), indicate that linear programming based pricing provides a good way to generate a good set of columns. An additional advantage of this approach is that the resulting set partitioning problem may be solved by more sophisticated techniques such as branch-and-cut. As far as we know, there do not exist methods that successfully combine column generation and cut generation techniques in the context of set partitioning.

Many of the ideas presented are applicable to all problems that can be formulated as a set partitioning problem with availability constraints, as is defined in Section 2. These problems are characterized by the fact that a set of tasks has to be assigned to a set of available resources, in such a way that all tasks assigned to a single resource can be feasibly processed by that resource. Because all our approximation algorithms for the pricing problem only modify the set of tasks currently assigned to a resource by adding and deleting tasks, it is relatively straightforward to modify the branch-and-price algorithm so that it can be used in other contexts. The problem specific characteristics relating to the cost and feasibility of assigning a set of tasks to a resource only have impact on the insertion function $I_r^k(j)$. However, a problem specific optimization algorithm for the pricing problem has to be developed, if the problem is to be solved to optimality.

		A_1				A_2			
Problem	Z_{LP}	CPU	nodes	Z_{BEST}	error	CPU	nodes	Z_{BEST}	error
DAR30.1	83681.0	61	7	83709	0.76	147	3	83687	0.16
DAR30.2	104755.0	19	1	104784	0.61	43	1	104755	0
DAR30.3	104189.0	8	1	104206	0.41	24	1	104189	0
DAR30.4	93988.3	54	13	93992	0.08	171	11	93999	0.25
DAR30.5	83978.0	13	1	84064	2.16	33	1	83978	0
DAR30.6	115235.0	16	1	115248	0.25	45	1	115235	0
DAR30.7	83668.0	14	1	83689	0.57	99	5	83673	0.14
DAR30.8	94101.0	11	1	94167	1.61	23	1	94101	0
DAR30.9	93887.0	15	1	93921	0.87	42	1	93887	0
DAR30.10	83724.0	16	1	83801	2.07	60	1	83724	0

Table 8: Performance of algorithms $A1$ and $A2$ for the problem class DAR30

References

[1] R. Anbil, R. Tanga and E.L. Johnson, A global approach to crew-pairing optimization, *IBM Systems Journal* **31**, 71-78 (1993).

[2] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh and P.H. Vance, *Branch-and-Price: Column Generation for Solving Integer Programs,* Report COC-94-03, Computational Optimization Center, Georgia Institute of Technology, Atlanta, Georgia (1994).

[3] G. Carpaneto and P. Toth, Some new branching and bounding criteria for the asymmetric travelling salesman problem, *Management Science* **26**, 736-743 (1980).

[4] M. Desrochers, J. Desrosiers and M. Solomon, A new optimization algorithm for the vehicle routing problem with time windows, *Operations Research* **42**, 342-354 (1992).

[5] Y. Dumas, J. Desrosiers and F. Soumis, The pickup and delivery problem with time windows, *European Journal of Operational Research* **54**, 7-22 (1991).

[6] G.A.P. Kindervater and M.W.P. Savelsbergh, *Local search in physical distribution management,* Memorandum COSOR 92-30, Eindhoven University of Technology (1992).

[7] G.L. Nemhauser, M.W.P. Savelsbergh and G.C. Sigismondi MINTO, a Mixed INTeger Optimizer, *Operations Research Letters*, to appear.

[8] D.M. Ryan and B.A. Foster, An integer programming approach to scheduling, *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling,* A. Wren (ed.), North-Holland, Amsterdam, 269-280 (1981).

[9] M.W.P. Savelsbergh, *A branch-and-price algorithm for the generalized assignment problem,* Report COC-93-02, Computational Optimization Center, Georgia Institute of Technology, Atlanta, Georgia (1993).

[10] M. SOL AND M.W.P. SAVELSBERGH, *The General Pickup and Delivery Problem*, COSOR Memorandum 92-44, Eindhoven University of Technology (1992); *Transportation Science*, to appear.

[11] P.H. VANCE, C. BARNHART, E.L. JOHNSON AND G.L. NEMHAUSER, *Solving binary cutting stock problems by column generation and branch and bound*, Report COC-92-09, Computational Optimization Center, Georgia Institute of Technology, Atlanta, Georgia (1992).