The Pickup and Delivery Problem with Time Windows and Precedences*

Mikkel Sigurd, David Pisinger, Michael Sig
Dept. of Computer Science, University of Copenhagen,
Universitetsparken 1, DK-2100 Copenhagen, Denmark

Abstract

In the classical Vehicle Routing Problem (VRP) we wish to service some geographically scattered customers with a given number of vehicles at the minimal cost. In the present paper we consider a variant of the VRP where the vehicles should deliver some goods between groups of customers. The customers have an associated time window, a precedence number and a quantity. Each vehicle should visit the customers within their time windows, in nonincreasing order of precedence and such that the capacity of the vehicle is respected. The problem will be denoted the Pickup and Delivery Problem with Time Windows and Precedency Constraints (PDPTWP). PDPTWP has applications in the transportation of live animals where veterinary rules demand that the livestocks are visited in a given sequence in order not to spread specific diseases.

A tight formulation based on Dantzig-Wolfe decomposition is proposed. The formulation splits the problem into a master problem, which is a kind of set-covering problem, and a subproblem which generates legal routes for a single vehicle. The LP-relaxation of the decomposed problem is solved through delayed column generation, and computational experiments show that the obtained bounds are less than 0.7% from optimum for practical problems.

Exact and heuristic algorithms for the subproblem are presented and their complexity is discussed. Moreover several branching strategies for reaching an integer solution are presented. The paper is concluded with a comprehensive computational study involving real-life instances from the transportation of live pigs.

1 Introduction

The SPF Association in Vejen, Denmark is responsible for planning transportation of live pigs in Denmark according to a number of veterinary restrictions. In order to avoid the

*Technical Report 00/08, DIKU, University of Copenhagen, DK-2100 Copenhagen spread of diseases, the transportations must be carried out such that each vehicle visits the livestocks according to a nonincreasing health code. At the end of each day, the vehicle is cleaned and thus it is disease-free the following day. The SPF company schedules around 300 transportations every day, using around 90 vehicles.

The problem may be seen as a Vehicle Routing Problem with Time Windows (VRPTW) since we have m vehicles to service n geographically scattered customers. The vehicles are located in a number of depots, and each vehicle has an associated capacity which may not be exceeded during the transportation. It is assumed that the vehicles leave the depot with no cargo, and arrive at the same depot wihout any cargo. Each customer has an associated time window in which he should be visited. If a vehicle arrives before the time window of a customer, it is assumed that the vehicle waits until the window opens.

The objective of the problem is to satisfy a number of deliveries at the lowest possible transportation cost. Each delivery consists of a seller (of live animals) and a buyer. Immediately after having picked up the animals at a seller, the vehicle should proceed to the corresponding buyer. Situations may however occur where a delivery consists of more than one seller to the same buyer, or more than one buyer to the same seller. In such a case it is allowed to first pick up the cargo from all the sellers before proceeding to the buyer (or vice versa). A compund delivery may however also be treated as a number of individual deliveries. Thus the problem may be seen as a kind of a Pickup and Delivery Problem with Time Windows (PDPTW).

The last restriction relates to the health-code of each livestock. In order not to spread diseases a vehicle should visit the customers according to a nonincreasing order of the health-code. This restriction imposes a precedency constraint to the problem which then becomes a Pickup and Delivery Problem with Time Windows and Precedency Constraints (PDPTWP).

To our knowledge PDPTWP has not previously been considered in the literature, but due to its similarity to VRPTW and PDPTW, several techniques from the latter problems may be used for its solution.

Kohl [Koh95] gives an overview of of exact solution methods for the VRPTW. As the problem is $\mathcal{NP}$-hard, all solution methods are based on branch-and-bound techniques using some kind of relaxation for deriving the lower bounds. A number of decomposition techniques have been proposed including *Lagrange relaxation*, *variable splitting* and *Dantzig-Wolfe decomposition*. Also *cutting plane* techniques are frequently used to tighten the formulation. Dumas, Desrosiers and Soumis [DDS91] solve a PDPTW through branch-and-bound and column generation. The authors describe how the associated graph can be reduced by some simple rules based on the time windows and capacity constraints. Moreover, a dynamic algorithm for generating routes is presented. Other algorithms for the route generation are presented in Sol and Savelsbergh [SS94] where two heuristic methods are described. The first heuristic is a kind of insertion algorithm which repeatedly inserts a customer in the route as long as the associated reduced costs are decreased. The second heuristic aims at improving existing columns. Sol and Savelsbergh propose two branching strategies for obtaining an integer solution. Ideally the branching strategy should keep the search tree balanced such that there is an equal progression in the solution effort at both subproblems. The first strategy divides the solution space according to whether or not a customer is serviced by a vehicle. The second strategy considers whether or not two customers are serviced by the same vehicle. In [SS95] Savelsbergh and Sol give an overview of the Pickup and Delivery Problem (PDP) in a general formulation. Several variants of PDP are discussed, and a number of exact and heuristic solution methods are presented. Bernhard et al. [BJN+96] discuss several aspects of using column generation with a branch-and-bound algorithm, including the solution of the pricing problem, branching strategies and derivation of additional constraints.

In the following we will present a graph formulation of PDPTWP. The problem is defined on an oriented graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where the node set $\mathcal{N}$ consists of customers $\mathcal{C}$ and depots $\mathcal{D}$. To each customer is associated a quantity q_i. A positive value of q_i means that the vehicle should pick up a load of the given size, and thus we will denote the corresponding customer a seller. A customer with a negative value of q_i is denoted a buyer. To each customer $i \in \mathcal{C}$ is assigned a time window $[a_i, b_i]$ and a precedency code p_i. A vehicle should visit the customers within their time window and according to nonincreasing precedency numbers.

The customers should be serviced by a set of vehicles $\mathcal{V}$. Each vehicle $k \in \mathcal{V}$ has an associated capacity Q_i, a start depot D_k^+ and an end depot D_k^-. In the following we will denote the set of start depots $\mathcal{D}^+ = \cup_{k \in \mathcal{V}} D_k^+$ and the set of end depots $\mathcal{D}^- = \cup_{k \in \mathcal{V}} D_k^-$. Every depot $d \in \mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$ has an associated time window $[a_d, b_d]$ within which it may be visited.

The cost of traveling from node i to node j is denoted c_{ij} and the corresponding travel time is assumed to be $t_{ij} > 0$. A typical choice of the cost function is the actual expenses (time and distance) for driving from i to j. If no edge exists between two nodes i and j, we may set the corresponding cost c_{ij} to infinity. If the number of vehicles used should be minimized one may add a large number to the cost of every edge leading from the start depot $\mathcal{D}^+$ to a customer.

The set of edges $\mathcal{E}$ form a complete graph on $\mathcal{C} \cup \mathcal{D}$. To ensure that the constraints corresponding to the deliveries are satisfied we remove all edges from a seller which do not lead to the corresponding buyer. In a similar way we remove all edges from a buyer which do not come from the corresponding buyer. In the case where two or more sellers deliver to the same buyer we maintain edges between the sellers. The same applies for the case where a seller delivers to a number of buyers.

The graph may be further pruned by removing all edges entering a start depot $\mathcal{D}^+$ and leaving an end depot $\mathcal{D}^-$. Edges between a start depot $\mathcal{D}^+$ leading to a buyer may also be removed, and a similar observation holds for edges going from a seller to an end depot $\mathcal{D}^-$. Edges between sellers and buyers which violate the precedency constraint may also be removed.

Finally, we may remove all edges that can not be part of any legal path because of time and capacity constraints. This is checked by generating a minimal path for each edge e containing e and checking if this path is legal. For an edge e between a start depot D_+ and a seller s with corresponding buyer b, the path $[D_+ \longrightarrow s \longrightarrow b \longrightarrow D_-]$ is considered. If this path is illegal, e can not be part of any legal path and can thus be removed from the graph. Edges between buyers and end depots are checked in a similar fashion. For edges that does not start or end in a depot, i.e. [*seller* $\longrightarrow$ *buyer*], [*seller* $\longrightarrow$ *seller*], and [*buyer* $\longrightarrow$ *seller*] a minimal path for each depot has to be checked. Only if all paths are illegal the edge can be removed from the graph.

To define PDPTWP as an ILP-problem we introduce three types of variables. Let x_{ijk} for $i, j \in \mathcal{N}, k \in \mathcal{V}$ be 1 iff vehicle k is driving from node i to node j. For a given vehicle k let s_{ik} for $i \in \mathcal{N}, k \in \mathcal{V}$ denote the time of arrival at node i (which can be a customer or a depot). Moreover let y_{ik} for $i \in \mathcal{N}, k \in \mathcal{V}$ be the used capacity of vehicle k when arriving at node i. If vehicle k does not visit node i we set $s_{ik} = 0$ and $y_{ik} = 0$. This leads to the following formulation

$$\text{minimize} \quad \sum_{k\in\mathcal{V}}\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} c_{ij}x_{ijk} \tag{1a}$$

$$\begin{aligned}
\text{subject to} \quad \sum_{k\in\mathcal{V}}\sum_{j\in\mathcal{N}} x_{ijk} &= 1, \ \forall i \in \mathcal{C} && \text{(1b)}\\
\sum_{j\in\mathcal{N}} x_{D_k^+jk} &= 1, \ \forall k \in \mathcal{V} && \text{(1c)}\\
\sum_{i\in\mathcal{N}} x_{iD_k^-k} &= 1, \ \forall k \in \mathcal{V} && \text{(1d)}\\
\sum_{i\in\mathcal{N}} x_{ihk} - \sum_{j\in\mathcal{N}} x_{hjk} &= 0, \ \forall h \in \mathcal{C}, \forall k \in \mathcal{V} && \text{(1e)}\\
y_{D_k^+k} &= 0, \ \forall k \in \mathcal{V} && \text{(1f)}\\
\sum_{i\in\mathcal{N}} Q_k x_{ijk} &\geq y_{jk}, \ \forall j \in \mathcal{N}, \forall k \in \mathcal{V} && \text{(1g)}\\
y_{ik} + x_{ijk}q_i - K(1 - x_{ijk}) &\leq y_{jk}, \ \forall i \in \mathcal{N}, \forall j \in \mathcal{N}, \forall k \in \mathcal{V} && \text{(1h)}\\
s_{ik} + t_{ij} - K(1 - x_{ijk}) &\leq s_{jk}, \ \forall i \in \mathcal{N}, \forall j \in \mathcal{N}, \forall k \in \mathcal{V} && \text{(1i)}\\
p_i - K(1 - x_{ijk}) &\leq p_j, \ \forall i \in \mathcal{C}, \forall j \in \mathcal{C}, \forall k \in \mathcal{V} && \text{(1j)}\\
a_i \leq s_{ik} &\leq b_i, \ \forall i \in \mathcal{N}, \forall k \in \mathcal{V} && \text{(1k)}\\
y_{ik} &\geq 0, \ y_{ik} \in \mathbb{R}, \forall i \in \mathcal{N}, \forall k \in \mathcal{V} && \text{(1l)}\\
s_{ik} &\geq 0, \ s_{ik} \in \mathbb{R}, \forall i \in \mathcal{N}, \forall k \in \mathcal{V} && \text{(1m)}\\
x_{ijk} &\in \{0, 1\}, \ \forall i \in \mathcal{N}, \forall j \in \mathcal{N}, \forall k \in \mathcal{V} && \text{(1n)}
\end{aligned}$$

Constraint (1b) ensures that each customer is visited exactly once. The constraints (1c) – (1d) ensures that each vehicle departs from its start depot and arrives at its end depot. The constraints (1e) ensures that if a vehicle arrives at a node then it must also depart from the node. As the arrival times s_{ik} of a vehicle will be strictly increasing, we are sure that no subtours will occur in the solution, and thus constraints (1c) – (1e) guarantee that each vehicle follows a consecutive path from its start depot to its end depot. The capacity constraints are ensured by the constraints (1f) – (1h), while the constraints on the time windows are ensured by constraints (1i) and (1k). Finally constraints (1j) ensures that the nodes are visited according to the precedency. The constant K is a sufficiently large number used to model conditional constraints. The model is an *edge formulation* of the problem since the decision variables x_{ijk} are connected to the use of a single edge.

The edge formulation of the problem has $O(n^2m)$ constraints and $O(n^2m)$ variables, where n is the number of nodes $|\mathcal{N}|$ and m is the number of vehicles $|\mathcal{V}|$. Due to the use of large constants K in the constraints (1h) – (1j) an optimal solution to the LP-relaxation may contain several fractional variables, and thus being far from an integer optimal solution. This also means that the lower bound obtained from LP-relaxation in general will be quite weak, and thus cannot be used to efficiently prune the search-tree of a branch-and-bound algorithm.

In the following section we will introduce a different formulation based on Dantzig-Wolfe decomposition where the corresponding LP-relaxation provides a tighter bound than by use of the edge formulation. This formulation will be denoted the *path formulation.* As the model may become exponentially large, we use delayed column generation to solve its LP-relaxation. Section 3 presents several algorithms for solving the pricing problem associated with the column generation. An exact branch-and-bound algorithm is finally developed in Section 4 which applies bounds from the LP-relaxation of the path formulation. The paper is concluded by numerous computational experiments with real-life data instances in Section 5. The largest instance solved to optimality within one hour has more than 400 nodes and 12 000 edges.

2 Dantzig-Wolfe decomposition

The idea in Dantzig-Wolfe decomposition is to express the solution space as a convex combination of subsolutions to some of the constraints. Thus let $\mathcal{P}$ be the set of legal paths in the problem. A path is legal if it begins in the depot associated to a given vehicle, visits a number of customers within their time windows, respecting the precedence and capacity rules, and returns back to the depot of the vehicle.

For a single path $p \in \mathcal{P}$ let c_p be the associated cost, and let $\delta_p^i = 1$ iff customer i is contained in the path. Moreover let $\sigma_p^k = 1$ iff vehicle k is used for the path p. Using the binary variable x_p to indicate whether path p is selected, we get the following *path-formulation*

$$\text{minimize} \quad \sum_{p \in \mathcal{P}} c_p x_p \tag{2a}$$

$$\text{subject to} \quad \sum_{p \in \mathcal{P}} x_p \delta_p^i \geq 1, \ \forall i \in \mathcal{C} \tag{2b}$$
$$\sum_{p \in \mathcal{P}} x_p \sigma_p^k \leq 1, \ \forall k \in \mathcal{V} \tag{2c}$$
$$x_p \in \{0,1\}, \ \forall p \in \mathcal{P}. \tag{2d}$$

The model will be denoted the *master problem* or just MP($\mathcal{P}$) for short. Constraint (2b) demands that each customer is visited, and constraint (2c) ensures that a vehicle is used no more than once.

Apart from being much simpler than the formulation (1) the new formulation has the advantage that it is very easy to controle the additional constraints as part of the generation of legal paths. Moreover, the new formulation leads to tighter lower bounds by LP-relaxation since every LP-solution to (2) corresponds to an LP-solution to (1) while the opposite is not the case in general. The path-formulation may however become very large, as the number of legal paths may be exponential in the input size. In order to get around this problem we will use delayed column generation for solving the LP-relaxation of (2).

Thus in each iteration we consider a *restricted master problem* defined on a subset $\mathcal{P}'$ of the paths $\mathcal{P}$. Let π_i be the dual variables associated with constraints (2b) and μ_k the dual variables associated with (2c) when solving MP($\mathcal{P}'$) to LP-optimality. The reduced cost of a path $p \in \mathcal{P}$ corresponding to vehicle k is given by

$$c_p^\pi = c_p - \sum_{i \in \mathcal{C}} \delta_p^i \pi_i - \mu_p^k \tag{3}$$

where $\delta_p^i = 1$ iff customer i is contained in the path. In order to extend the current set $\mathcal{P}'$ of paths according to the Dantzig rule of the Simplex algorithm, the path $p \in \mathcal{P}$ with largest negative reduced cost c_p^π should be found. This problem may be recognised as finding a shortest path between the start depot D_k^+ and the end depot D_k^- of a vehicle k. The cost c'_{ij} associated to a given edge (i,j) equals the original edge cost c_{ij} minus the dual variable π_j associated with node j. The found path must be legal, i.e. it must satisfy the constraints on the capacity, time windows and precedency. These additional constraints make the shortest path problem become $\mathcal{NP}$-hard.

Initially we may choose the subset $\mathcal{P}'$ as a set of paths with very large associated costs, each path visiting a single customer. In each of the following iterations we solve the restricted master problem MP($\mathcal{P}'$) to LP-optimality obtaining the dual variables. These are then used to find a new legal path $p \in \mathcal{P}$ with largest negative reduced cost (3). We add this path p to $\mathcal{P}'$ and repeat the process until a legal path with negative reduced costs cannot be found. In this case we get from weak duality [Chv83] that an optimal LP-solution to MP($\mathcal{P}$) has been found.

3 Solving the pricing problem

The *pricing problem* is the problem of finding the legal path $p \in \mathcal{P} \setminus \mathcal{P}'$ with largest negative reduced cost c_p^π corresponding to the current restricted master problem $\mathrm{MP}(\mathcal{P}')$. This problem must be solved each time a column is added during the column generation. Because of the constraints on the legal paths, the problem is $\mathcal{NP}$-hard, which means that solving the pricing problem for every column added during column generation is very time consuming. Thus we used the following strategy:

- Instead of finding the path with largest negative reduced cost, we just find some path with negative reduced cost. In many cases this can be done using a fast heuristic algorithm. Only if we are not able to find any paths with negative reduced cost using the heuristic, we solve the pricing problem using an exact algorithm.
- Often our heuristic and exact algorithms find several paths with negative reduced cost. Instead of adding just one path, we add l paths. This greatly reduces the number of times we need to solve the pricing problem, but also makes the restricted master problem $\mathrm{MP}(\mathcal{P}')$ larger than necessary.

We have implemented a heuristic and two exact pricing algorithms. We have also implemented several relaxations of the exact algorithms to obtain more heuristic algorithms.

3.1 Construction algorithm

The *construction algorithm* has been proposed by Sol and Savelsbergh in [SS94]. The algorithm starts out with an empty path and performs a number of greedy iterations. In each iteration the customer that decreases the reduced cost the most is added to the path in the best place on the path. The construction algorithm is a non-optimal since the greedy choices made in each iteration do not necessarily result in an optimal path. The running time of the construction algorithm is $O(nh^2)$, where n is the number of customers, and h is an upper bound (independent of n) on the number of customers visited in a single path. An upper limit h on the number of customers visited is reasonable for the considered problem, since loading and unloading of live animals is so time-consuming, that a vehicle only will be able to visit a few customers every day.

3.2 Depth-first search algorithm

The *depth-first search algorithm* develops all legal paths for the problem at hand during a depth-first search of the graph. Illegal paths are cut off during the search of the graph by checking time, precendence, and capacity constraints whenever a node is added to a path. Since the depth-first search algorithm finds all legal paths it is possible to add more than one path with negative reduced cost to the master problem. The depth-first search algorithm has complexity $O(n!)$ in worst case, but if we assume that the length of every legal path is less than k, the complexity becomes $O(n^k)$.

3.3 State algorithm

The *state algorithm* was proposed by Dumas, Desrosiers and Soumis in [DDS91] for the PDPTW. We have adapted the algorithm for solving the pricing problem for the PDPTWP.

The state algorithms is based on the idea of Dijkstra's shortest path algorithm, only another dominance criteria is used since the constraints on the legal paths makes Dijkstra's dominance criteria invalid. The state algorithm maintains a set of states $S(j, M, T, R)$ each corresponding to a path in the graph, where j is a node, M is the set of nodes previously visited on the path, T is the arrival time of the path to node j, and R is the reduced cost of the path. Like the depth-first search algorithm the set of states is initially just one state corresponding to the empty path starting in a start depot. In each iteration of the state algorithm a state $S(j, M, T, R)$ is picked out and the corresponding route is extended to each of the neighbours of j thus creating a number of new states which are added to the set of states. Only states corresponding to legal paths are added. If the end depot is visited a complete legal path has been constructed and if its reduced cost c_p^π is negative it can be added to the restricted master problem MP($\mathcal{P}'$).

If we have two states $S_1(j, M_1, T_1, R_1)$ and $S_2(j, M_2, T_2, R_2)$ corresponding to the same node j and if $T_1 \leq T_2,\ R_1 \leq R_2,\ M_1 = M_2$ we say that state S_1 *dominates* S_2. This is the case if the paths corresponding to the two states have visited the same customers in different order and if the path corresponding to S_1 have visited the customers in a way that results in a faster path with lower reduced cost than the path corresponding to S_2. If this is the case all paths that are extended from state S_2 will not be better than the similar paths extended from S_1, which means that we can delete S_2. This way we cut off bad paths at an early time and only spend time extending good paths. In proposition 1 we have proved that this dominance criteria is valid.

Proposition 1 (Dominance) *For two states $S_1(j, M_1, T_1, R_1)$ and $S_2(j, M_2, T_2, R_2)$ corresponding to the same node j, the dominance criteria $T_1 \leq T_2,\ R_1 \leq R_2,\ M_1 = M_2$ is valid for PDPTWP.*

Proof: Assume we have two states $S_1(j, M_1, T_1, R_1)$ and $S_2(j, M_2, T_2, R_2)$ in a node j and that $T_1 \leq T_2,\ R_1 \leq R_2,\ M_1 = M_2$. Let r_1 be the path corresponding to state S_1 and r_2 be the path corresponding to state S_2. Let i be a neighbour of j so that r_2 can be extended to i legally, i.e. i can be visited within its time window, i has lower precedence than j, and the capacity constraints are not violated. By extending r_2 to i we obtain a new state $S_2^i(i, M_2 \cup \{i\}, T_2 + t_{ji}, R_2 + r_{ji})$ for the route r_2^i obtained by extending r_2 to i.

Obviously r_1 can also be extended to i and we thereby obtain another route r_1^i and a new state $S_1^i(i, M_1 \cup \{i\}, T_1 + t_{ji}, R_1 + r_{ji})$ corresponding to r_1^i. Since $T_1 + t_{ji} \leq T_2 + t_{ji}$, $R_1 + r_{ji} \leq R_2 + r_{ji}$ and $M_1 \cup \{i\} = M_2 \cup \{i\}$ state S_1^i dominates S_2^i. This means that any extension of S_2 will always be dominated by the similar extension of S_1, and thus we can avoid to further extend S_2. $\square$

In worst case the state algorithm will not be able to make any dominations which means that all possible states will have to be considered. The state algorithm will then be similar to the label algorithm which has complexity $O(n^h)$ if we assume that the length of any legal paths is less than a constant h.

3.4 Heuristics

The depth-first search algorithm will consider all legal paths, which makes the algorithm very time consuming. If we can cut off some paths that look unpromising at an early stage we can improve the execution time of the algorithm at the expense of the algorithm no longer being exact. We have experimented with the following rules:

- Customers located far apart: We cut off paths that visit customers far from each other since good paths will probably visit customers located close to each other in order to service most possible deliveries.
- High reduced cost: We cut off paths that have a high reduced cost during the execution of the depth-first search algorithm, assuming that these paths cannot be extended to paths with negative reduced cost.

These rules of thomb can also be used for the state algorithm thus making the state algorithm a heuristic. If we relax the dominance criteria in the state algorithm we will be able make more dominations which will improve the execution time of the state algorithm. The state algorithm will then be a heuristic since we may cut off some paths that can be extended into good paths. We have experimented with the two heuristic rules above and several relaxed dominance criterias.

4 Exact solution through branch-and-bound

The LP-solutions to MP($\mathcal{P}$) found by column generation is not necessarily integral. In order to obtain integral solutions we apply a branching rule which excludes the fractional solutions. When designing the branching scheme we have made the following design goals:

- The branching scheme should divide the solution space evenly since this will exclude a large amount of solutions in each of the subproblems, which means that we have good chance of decreasing the gap between the best fractional solution and the best integral solution in each node of the branching tree.
- The branching scheme should help the pricing problem, i.e. the pricing algorithms should be able to make use of the fact that they are working on a smaller solution space to run faster. This design goal is as much a desing goal for the pricing algorithms as it is for the branching rule.

We have implemented two branching schemes; branching on routes and branching on assignment of customers to vehicles.

4.1 Branching on routes

Branching on routes divides the solution space into two subspaces. In one subspace a route $p_0 \in \mathcal{P}$ is required to be part of the solution and in the other space p_0 is excluded from the solution. Since every route $p \in \mathcal{P}$ is represented in the path formulation by a decision variable x_p this branching scheme divides the solution space by adding the constraints $x_{p_0} = 1$ and $x_{p_0} = 0$ respectively.

Unfortunately this branching scheme does not divide the solution space evenly since requiring that $x_{p_0} = 1$ is much stronger than requiring that $x_{p_0} = 0$. In the case of setting $x_{p_0} = 1$ the vehicle and the customers on the route have been assigned to a route which means that all solutions where either the assigned vehicle or the assigned customers are assigned to other routes can be excluded from the solution space. In the case of $x_{p_0} = 0$ we can only exclude solutions which p_0 is part of.

We can compensate for the uneven division by branching on the highest fractional variable $x_{p_0} < 1$. This way we branch on a variable that is most likely to be set to 1 in an integral solution. This strengthens the branching on $x_{p_0} = 0$ since the fractional solution is changed most possible. Our experiments have shown that branching on routes along with this rule for selecting the branching variable results in a fairly even division of the solution space.

When branching on routes we must be able to find all paths with negative reduced cost when solving the pricing problem since the path with lowest reduced cost may be forbidden by a branching constraint. This means that we must use the depth-first search algorithm as the exact algorithm when using this branching strategy since the state algorithm is only guaranteed to find the path with lowest reduced cost. Of course we can still use the state algorithm as a heuristic.

The state and depth-first search algorithms can only make use of the case of $x_{p_0} = 1$ in which case we omit route generation for the vehicle belonging to p_0 and cut off all routes visiting the customers visited on p_0. In the case of $p_0 = 0$ we can not utilize the constraint during route generation, in fact we have to make sure that p_0 is not added to the path formulation again if it is generated during the route generation.

4.2 Branching on assignment of customers to vehicles

In this branching scheme the solution space is divided into two subspaces by requiring in one subspace that a customer $i_0 \in \mathcal{C}$ must be visited by a vehicle $k_0 \in \mathcal{V}$ and in the other subspace requiring that c_0 must not be visited by v_0. The branching is applied by adding the constraints $\sum_{p \in \mathcal{P}} x_p \delta_p^{i_0} \sigma_p^{k_0} = 1$ and $\sum_{p \in \mathcal{P}} x_p \delta_p^{i_0} \sigma_p^{k_0} = 0$ respectively to the path formulation.

As was the case of branching on routes, assigning a customer to a vehicle is a stronger constraint than forbidding a customer from being visited by a vehicle. Again this unevenly division of the solution space can be repaired by branching on the vehicle and a customer belonging to the highest fractional variable. Our experiments show that branching on assignment of customers to vehicles together with this selection strategy results in a evenly division of the solution space.

The pricing algorithms can make use of both constraints when generating routes. In the case of customer i_0 being assigned to vehicle k_0 we can cut off all routes for vehicle k_0 that do not visit i_0. This can be done for routes belonging to k_0 in the state and depth-first search algorithms during the route generation by checking in each step if it is still possible to add i_0 to the route. If not, the route generation can cut off the further extension of this route.

In the case of forbidding the assignment of customer i_0 to vehicle k_0 we can make use of this in the route generation algorithms by cutting of routes belonging to k_0 if they visit i_0. This can be done during the route generation, which means that further extension of such routes is avoided.

5 Computational experiments

We have implemented a branch and price algorithm to solve PDPTWP including the three route generation algorithms and the two branching schemes. To help our implementation we have used ABACUS ("A Branch-And-CUt System") [Thi95] which is a collection of C++ classes, that significantly reduceds the work of implementing branch and bound like algorithms. Our experience with ABACUS have been very positive and it has helped us to test several strategies with limited programming effort. ABACUS provides an interface to CPLEX [Cpl95] and SoPlex [Wun97], which we have used to solve the linear programs resulting from the column generation. All test has been carried out on an AMD K6 200 MHz.

We have tested several different configurations of the branch and price algorithms in order to find the best exact algorithm and some good heuristic algorithms as well. In all

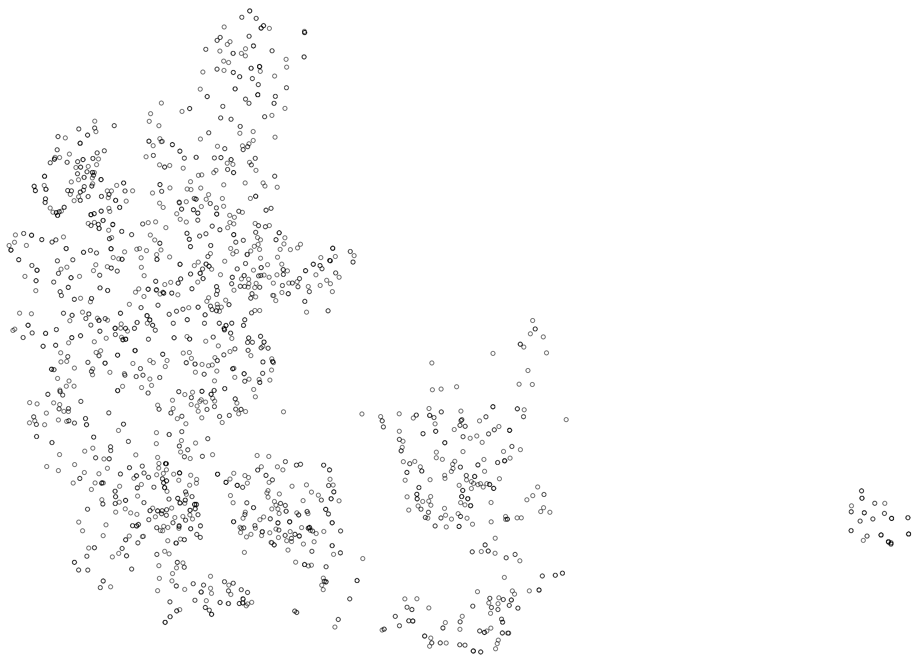

Figure 1: The figure shows the geographical location of all sellers and buyers serviced by the SPF Association in week 50, 1999

cases we started the column generation by using the construction algorithm, since this algorithm is very fast and generates paths of high quality. Whether the branch and price algorithm is exact or heuristic depends on whether we use an exact algorithm to solve the pricing problem. If anly heuristic algorithms are applied for the pricing problem, the delayed column generation may terminate premature and thus not provide a legal lower bound. This may imply that some parts of the branch-and-bound tree are pruned although they contain the the optimal solution.

To test our algorithm we have constructed a series of subproblems from the data set supplied by the SPF Association, which is a real life data set of about 1300 transportation request to be handled by 90 vehicles in five days. The complete data set is illustrated in Figure 1. In table 1 we have shown the characteristics of the 17 smaller test problems we have used.

Unfortunately the data set provided by the SPF Association only has very few time windows (less than 10% of the customers) and the present time windows are often very

problem	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
nodes	68	68	64	62	56	66	60	70	62	60	58	62	54	54	64	62	66
edges	740	765	583	554	484	632	494	749	561	518	494	644	481	423	596	608	704

Table 1: The table show the size of the representing graph of the 17 test problems.

wide. In order to test the performance of the algorithm on problems with more tight time windows we have created 130 other problems with time windows. Our experiments on problems with time windows will be described in section 5.3.

5.1 Exact algorithms

By using either the state algorithm with branching on customers and vehicles or the depth-first search algorithm together with either branching strategy to solve the pricing problem we obtain an exact algorithm. In table 2 we have shown the computation time for the 4 configuration obtained by varying the pricing algorithm, the heap order in the state algorithm, and the branching strategy for the depth-first search algorithm. Since the CPU time varies considerably from problem to problem it is not a correct measure of the algorithms performance. In the column *Performance* we have calculated the average deviation from the mean CPU time for each configuration. The column *Variance* shows the variance on the performance.

Algorithm	Branching	Heap order	CPU	Performance	Variance
State	C. and v.	Red. cost	1:34	-2,30%	6,71%
State	C. and v.	Travel time	1:17	0,34%	7,86%
Depth-first	C. and v.	-	1:20	1,35%	14,56%
Depth-first	Routes	-	1:41	0,61%	3,07%

Table 2: The table shows the average performance of the exact configurations on the 17 test problems.

In table 3 each of the parameters are compared separately. We see that there is hardly any difference whether we use the state algorithm together with branching on customers and vehicles or we use the depth-first search algorithm together with either branching strategy. We would expect branching on customers and vehicles to perform better since this strategy should divide the solution space more evenly, but an examination of the branching trees of the two branching strategies shows that both strategies divide the solution space quite evenly.

Comparison	Performance	Variance
Branching on customer and vehicles	-0,45%	5,88%
Branching on routes	1,35%	14,56%
Heap ordered by reduced cost	-2,30%	6,71%
Heap ordered by travel time	0,34%	7,86%
State algorithm	-0,98%	7,28%
Depth-first search algorithm	0,98%	8,82%

Table 3: The table shows a comparison of three parameters of the exact algorithms.

Branching	Heap order	Dom.	Dev.	Solved prob.	Perform.	Variance
C. og v.	Red. cost	RT	0,00%	17	-34,12%	11,26%
C. og v.	Red. cost	R	1,68%	15	-50,29%	27,67%
C. og v.	Red. cost	T	5,92%	10	-64,26%	16,23%
C. og v.	Time	RT	0,00%	17	-24,34%	12,97%
C. og v.	Time	R	0,20%	15	-56,49%	15,00%
C. og v.	Time	T	5,92%	10	-64,26%	16,23%
Routes	Red. cost	RT	0,00%	17	-41,94%	4,69%
Routes	Red. cost	R	1,11%	14	-65,27%	19,15%
Routes	Red. cost	T	5,79%	10	-78,72%	3,62%
Routes	Time	RT	0,00%	17	-44,82%	6,00%
Routes	Time	R	0,24%	15	-63,84%	7,49%
Routes	Time	T	5,79%	10	-78,98%	3,66%

Table 4: The table shows the performance and the quality of solutions of the heuristic algorithms based on relaxed dominance criteria state algorithm.

5.2 Heuristics

By relaxing the state algorithm's dominance criteria we can obtain a heuristic pricing algorithm. We have experimented with three relaxed dominance criterias:

- **RT**: State S_2 is dominated by S_1 if $T_1 \leq T_2,\ R_1 \leq R_2$.
- **R**: State S_2 is dominated by S_1 if $R_1 \leq R_2$.
- **T**: State S_2 is dominated by S_1 if $T_1 \leq T_2$.

In table 4 the performance of the branch and price algorithm using heuristic state algorithms obtained by relaxing the dominance criteria is shown. The performance is calculated relatively to the performance of the best exact algorithm. In some cases the solution found by the heuristic algorithms contain dummy variables, which means that the heuristic has not been able to find a valid integral solution. In these cases we consider the problem as unsolved. In table 4 we have shown the number of solved problems and the average deviation from the optimal value for the solved problems.

In table 5 each parameter of table 4 is compared. We see that branching on routes performs slightly better than branching on assignment of customers to vehicles and that the deviation and the number of solved problems are about the same for the two branching strategies. The most interesting is the comparison of the three dominance criterias. We see that by using dominance criteria RT we are able to reach optimal solutions 36% faster than by the exact algorithm. The two other dominance criteria solve the problems somewhat faster but do not find an optimal solution for all the considered problems.

Comparison	Deviation	Solved problems	Perform.	Variance
Branching on c. and v.	2,29%	14,0	-48,96%	16,56%
Branching on routes	2,15%	13,8	-62,26%	7,43%
Heap ordered by red. cost	2,42%	13,8	-55,77%	13,77%
Heap ordered by time	2,02%	14,0	-55,46%	10,22%
Dominance criteria RT	0,00%	17,0	-36,31%	8,73%
Dominance criteria R	0,81%	14,8	-58,97%	17,33%
Dominance criteria T	5,85%	10,0	-71,56%	9,93%

Table 5: The table shows a comparison of three parameters for the branch and price algorithm using an heuristic state algorithm as pricing algorithm.

5.3 Time window problems

As previously mentioned the data set supplied by the SPF Association contained very few time windows. In order to test the performance of the branch and price algorithm on problems with time windows we have added time windows to 10 subproblems of the data set from the SPF Association. The characteristics of the 10 problems are shown in table 6.

We have varied the number of time windows and the size of the time windows which in all gives os 130 new problems for our testing. The performance of the best exact algorithm and the algorithm using the state heristic with dominance criteria RT as pricing algorithm on the problems with time windows are shown in table 7 and 8.

As seen from the CPU times in table 7 the performance of both algorithms improve dramatically when the test problems contain time windows. The average CPU time used by the exact algorithm to solve the test problems drops from a little under 17 minutes to just 5 seconds when we add 1 hour time windows to all customers. The performance of the heuristic algorithm is about 50% better than the exact algorithm and all problems but one have been solved to optimality.

5.4 Summary fo the computational experiments

When introducing the path formulation of PDPTWP we claimed that the LP relaxation would provide tight bounds on the optimal integral solutions. Our experiments showed that the LP-solution to MP($\mathcal{P}$) on average was 0.70% from the optimal solution at the root node of the branch and bound tree.

Problem	1	2	3	4	5	6	7	8	9	10
Nodes	74	94	80	76	74	88	114	104	106	98
Edges	839	1451	979	986	775	1215	1925	1713	1645	1308

Table 6: The table shows the 10 problems we have used to create the time window problems.

State algorithm with dominance criteria RTM			
Time windows			
Percentage	Size	Deviation	CPU
0%	-	0,00%	16:44
25%	4 hours	0,00%	12:41
25%	2 hours	0,00%	13:42
25%	1 hour	0,00%	5:31
50%	4 hours	0,00%	4:51
50%	2 hours	0,00%	4:25
50%	1 hour	0,00%	1:09
75%	4 hours	0,00%	2:23
75%	2 hours	0,00%	1:22
75%	1 hour	0,00%	0:10
100%	4 hours	0,00%	2:34
100%	2 hours	0,00%	0:23
100%	1 hour	0,00%	0:05

Table 7: The table shows performance of the branch and price algorithm using the exact state algorithm as pricing algorithm on the test problems with time windows.

State heuristic with dominance criteria RT					
Time windows			Solved		
Percentage	Size	Deviation	problems	Performance	Variance
0%	-	0,00%	10	-54,55%	5,33%
25%	4 hours	0,00%	10	-58,78%	2,68%
25%	2 hours	0,00%	10	-58,02%	2,48%
25%	1 hour	0,00%	10	-55,81%	1,25%
50%	4 hours	0,00%	10	-50,93%	4,84%
50%	2 hours	0,00%	10	-47,45%	4,88%
50%	1 hour	0,01%	10	-37,94%	6,12%
75%	4 hours	0,00%	10	-54,66%	3,25%
75%	2 hours	0,00%	10	-33,91%	6,89%
75%	1 hour	0,00%	10	-26,42%	10,72%
100%	4 hours	0,00%	10	-48,98%	2,44%
100%	2 hours	0,00%	10	-31,95%	9,53%
100%	1 hour	0,00%	10	-24,02%	12,61%

Table 8: The table shows performance of the branch and price algorithm using the state heuristic with dominance criteria RT as pricing algorithm on the test problems with time windows.

From our experiments we have found that there is only little difference between the performance of the two branching strategies. Also the performance of the two exact pricing algorithms are quite even. We have chosen the configuration branching on customers and vehicles together with the exact state algorithm to solve some larger subproblems of the data set supplied by the SPF Association with time windows added. The largest problem we were able to solve in less than an hour contained 410 nodes and 12872 edges and it was solved in 13 minutes and 39 seconds.

6 Conclusion

We have presented a new variant of the classical VRP, which is motivated by the transportation of live animals according to some veterinary constraints. The VRPTWP deals with time windows, capacity constraints, and precedency constraints. Since the additional constraints may be difficult to express in an MIP-model a path-formulation has been presented. The LP-relaxation of the path-formulation is solved through delayed column generation, providing relatively tight lower bounds for an exact algorithm based on branch-and-bound. The computational experiments with real-life problems show that the smaller problems may be solved to optimality in reasonable time. For the larger problems the deviation of a lower bound through column generation is computationally too demanding, and thus a relaxed version of the pricing problem has been applied. Due to the relaxation of the pricing problem we cannot guarantee that the found solutions are optimal, but they represent heuristic solutions of high quality.

The contribution of this paper can be summed up as follows: We have presented a new routing problem with applications in transportation of live animals and adapted existing techniques from VRP and PDP to develop an exact algorithm based on branch and price. In particular the algorithms for solving the pricing problem are new, and their application in a hierarchy according to solution time and quality is promising. A very thorough experimental work has been presented which made it possible to identify the most promising configurations of an exact and a heuristic algorithm. The largest problem solved to optimality involves more than 400 nodes and 12 000 edges which even for problems without precedency constraints is a very difficult task. For comparison Kallehauge [Kal00] reports the best solution times for Solomon test instances with up to 200 customers.

References

[BJN+96] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. School of Industrial and Systems Engineering, Georgia Institute of Technology, 1996.

[Chv83] Vašek Chvátal. *Linear Programming.* Freeman, 1983.

[Cpl95] Cplex. *Using the Cplex Callable Library.* Cplex Optimization, Inc., 1995.

[DDS91] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54:7–22, 1991.

[Kal00] Brian Kallehauge. Lagrange dualitet og ikke-differentiabel optimering anvendt i rutelægning (Danish). Technical Report, IMM-EKS-2000-13, DTU, Lyngby, Denmark, 2000.

[Koh95] Niklas Kohl. *Exact Methods for Time Constrained Routing and Related Scheduling Problems.* PhD thesis, Department of Mathematical Modelling, Technical University of Denmark, 1995.

[SS94] M. Sol and Martin W. P. Savelsbergh. A branch-and-price algorithm for the pickup and delivery problem with time windows. COSOR Memorandum 94-22, Eindhoven University of Technology, 1994.

[SS95] M.W.P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29:17–29, 1995.

[Thi95] S. Thienel. *ABACUS — A Branch-And-CUt System.* PhD thesis, Universität zu Köln, 1995.

[Wun97] Roland Wunderling. Soplex, the sequential object-oriented simplex class library. http://www.zib.de/Optimization/Software/Soplex/, 1997.