TWEAKER :

An Ant Colony Algorithm for solving the Time Window Assignment Vehicle Routing Problem with Uncertain Demand

September 25, 2012

Tim Lamballais Tessensohn
321789 TL

Abstract

This thesis considers the Time Window Assignment Vehicle Routing Problem (TWAVRP) with uncertain demand. This problem extends the well known Vehicle Routing Problem with Time Windows (VRPTW) by including demand scenarios and by requiring the placement of an additional time window within the ordinary time windows. A demand scenario specifies for each customer a realization of demand. The additional time window must be placed such that expected costs across demand scenarios are minimized. An Ant Colony algorithm is developed that can solve this problem for medium to large instances within a reasonable computation time.

Keywords: Time Windows, Vehicle Routing, Ant Colony Optimization

Supervisor: dr. A.F. Gabor
Co-reader: R. Spliet MSc

1 Introduction

Distribution networks pose countless intriguing challenges and numerous algorithms and techniques have been conceived to tackle these challenges exactly or approximately. One such challenge is the famous Vehicle Routing Problem with Time Windows (VRPTW) where multiple vehicles have to visit and service each customer within a specific time window under capacity constraints. This problem is well studied because of its practical relevance. Solomon (1987) created several benchmark instances for which the exact solutions and the best approximate solutions from heuristics are publicly available[1]. One limitation of the VRPTW is that customer demand is assumed fixed and only one routing schedule is created. This can cause problems if customer demand varies, as very different routes may be optimal for very different realizations of demand across customers. Arrival times at one customer could consequently also differ quite widely from delivery to delivery. However, customers typically want their deliveries to fall within a pre-specified time span to accommodate their own personnel scheduling for the processing of deliveries. This poses a question: when should this additional time window begin?
To solve this additional complication, endogenous time windows are distinguished from exogenous time windows. The original time windows of the VRPTW are the exogenous time windows and define the time span within which arrivals at a customer are allowed to occur. The endogenous time window of a customer is the time window within which deliveries at that customer should arrive. The endogenous time window must be placed within the exogenous time window for each customer and arrivals at a customer have to fall within the endogenous time window. By a priori specifying multiple realizations of demand the assignment of the endogenous time windows can be optimized in such a way that the expected costs of the routes for the demand realizations is minimal. Thus a new challenge is born, the Time Window Assignment Vehicle Routing Problem (TWAVRP).
Being able to solve the TWAVRP allows a supplier to ask its customers for a time interval within which deliveries can arrive and to return to its customers a time interval within which deliveries will arrive. An exact method to solve the TWAVRP is provided by Spliet and Gabor (2012). They use a Branch and Price procedure that can solve small instances of the TWAVRP. The aim of the current thesis is to develop an Ant Colony Optimization (ACO) heuristic that can solve medium to large instances within a reasonable computation time. The structure of this thesis is as follows: Section 2 describes the TWAVRP. This is followed by a discussion in section 3 of how the ACO meta-heuristic has been used to solve other VRP(TW) problems. Section 4 provides a general outline of TWEAKER, the heuristic approach for solving the TWAVRP. Section 5 shows how pheromones are constructed for the TWAVRP and sections 6 and 7 describe how the pheromones are used to generate route schedules and time window assignments. Section 8 provides numerical results and section 9 concludes.

2 Problem Description

Consider a graph $G = (V, E)$, with the set $V = \{0, \ldots, n\}$ of n customers with the depot as customer 0 and the set Ω of demand scenarios. A demand scenario $\omega \in \Omega$ is defined as a realization of demand for every customer. This demand is d_v^ω for a specific customer $v \in V \backslash \{0\}$ in a specific demand scenario $\omega \in \Omega$. A demand scenario ω happens with a probability p_ω. Vehicle capacity is denoted with Q and it must hold that $0 < d_v^\omega < Q$. Customer v has a service time s_v which is the time required for loading and unloading.
Each customer v has a pre-specified exogenous time window $[y_v^s, y_v^e]$. For each customer, all arrivals across demand scenarios must occur within its endogenous time window. The endogenous time window starts at y_v and has given width w_v, so $[y_v, y_v + w_v] \subset [y_v^s, y_v^e]$. A Time Window Assignment (TWA) is denoted with y and defines for every customer $v \in V$ an endogenous time

[1] http://w.cba.neu.edu/ msolomon/problems.htm

window $[y_v, y_v + w_v]$.
Let $c_{v,w}$ be the travel cost and $t_{v,w}$ the travel time for visiting a customer w after customer v. The travel cost $c_{v,w}$ is taken proportional to the Euclidean distance between v and w and $t_{v,w} = c_{v,w} + s_v$. A route schedule r is a path in G starting and ending in the depot that visits all customers. The depot can appear multiple times in r. A route schedule $r_\omega(y)$ for demand scenario ω has associated costs $c_{r_\omega(y)}$ and arrival times, $t^v_{r_\omega(y)}$ at customer v. Waiting at a customer is allowed, which means that going from customer v to customer w it can happen that $t^v_{r_\omega(y)} + t_{v,w} \leq t^w_{r_\omega(y)}$. The schedule $r_\omega(y)$ is considered feasible under y if capacity never exceeds Q when customer demand equals d^ω_v and if arrival times lie within y, $t^v_{r_\omega(y)} \in [y_v, y_v + w_v], \forall v \in V \backslash \{0\}$.
Let $r^*_\omega(y)$ be the feasible route schedule with lowest travel cost $c^*_{r_\omega(y)}$ for demand scenario ω and TWA y. The solution of the TWAVRP is the assignment y^* which minimizes expected costs across demand scenarios:

$$\sum_{\omega \in \Omega} p_\omega c^*_{r_\omega(y^*)} \leq \sum_{\omega \in \Omega} p_\omega c^*_{r_\omega(y)} \quad \forall y \tag{1}$$

Note that for different demand scenarios the optimal route schedules may be quite different. The assignment y^* can be such that for a demand scenario with a low probability to occur the best route schedule can have very high costs compared to those of other demand scenarios.

3 ACO and Time Windows

A popular method for constructing feasible routes r in VRPTW problems is the Ant Colony Optimization (ACO) meta-heuristic. ACO is inspired by the behavior of foraging ants discovering shortest trails to food sources. They leave a substance called 'pheromone' on the ground. The scent of pheromone differs across ant colonies, which helps ants to recognize trails from their fellow colony members. Ants create trails by randomly walking around until they find a food source, after which they travel back along their trail to the colony. Trails can lead past several food sources. The shortest trails are traversed fastest and therefore most often, which augments the scent. This provides positive feedback that guides the ants. Yet the ants choice is probabilistic, the intensity of pheromone along different trails only influences the probabilities for choosing a certain trail. The ants can choose to follow an existing trail or to make a new trail and explore new areas. If an ant finds a trail to a food source and that trail is closer to the colony than other trails, the positive feedback helps the other ants to understand that it is a short trail. As time passes pheromone evaporates. This means that trails that are longer or used less often than other trails will have a decreasing number of pheromones and therefore a lower probability to be picked as time passes. Evaporation therefore allows the colony to effectively stop using a trail.
In the ACO meta-heuristic, trails correspond to solutions and the length of the trail correspond to a solution quality that must be optimized. Pheromones are defined for solutions or parts of solutions and indicate the attractiveness of using that solution (part). To complement this, ACO algorithms include a visibility component that provides additional information on attractive features. This general set-up for pheromone and visibility allows the flexibility needed to adapt an implementation of ACO to suit a specific problem. ACO implementations cover a wide range of problems in Operations Research, see Dorigo and Stützle (2004) for several problems where ant algorithms have been successful.
The ACO algorithm can be implemented for a Vehicle Routing Problem (VRP) by letting a trail consist of arcs between customer locations. A trail consisting of arcs that start and end at the depot equals a route in the VRP. The customers would coincide with the food sources along this trail, with the demand of a customer corresponding to the size of a food source. The vehicle capacity in the VRP conforms with the maximum amount that an ant can carry from

a food source. Pheromones lie on the arcs between customer locations, where pheromones on arc (v, w) between customers v and w measure how attractive ants have found it in the past to visit w after v. In the VRP the visibility component for an arc (v, w) is commonly set as one divided by the distance between customer v and w. The ants move from location to location using probabilities based on the pheromones on and the visibility of the arcs. After each ant has visited each location, pheromones are increased based on the length of the routes the ants walked and pheromones are decreased according to an evaporation rate.
Rizzoli et al (2007) discuss ACO implementations that address variants of the VRP. One of the first and most competitive implementations for the VRPTW is the MACS-VRPTW algorithm by Gambardella et al (1999). This algorithm features two independent ant colonies, one whose aim is to minimize the number of vehicles used and another that pursues travel time minimization. The two colonies communicate by sharing the best solution. Since the introduction of MACS-VRPTW, ACO algorithms have been applied to variants of the VRPTW such as the VRPTW with backhauls (Reimann and Ulrich, 2006), a VRPTW with double time windows (Jia et al 2012) and a VRPTW with additional delivery men (Pureza et al 2012). These algorithms show that ACO can be used in situations where the VRPTW has become a sub-problem.
Ant algorithms can be combined with other meta-heuristics to help it escape from local optima. Yu et al (2011) propose a hybrid algorithm that combines ACO with tabu search and Dengiz et al (2010) combine ACO with Simulated Annealing. Another way to enhance the performance of ACO algorithms is local search (Gambardella et al 2012). The ACO algorithms for the VRPTW use either the 2-opt or Or-opt procedure of Potvin and Rousseau (1995) or the more general CROSS exchange described by Taillard et al (1997).
The TWAVRP is a variant of the VRPTW where the VRPTW has become a sub-problem. A heuristic that attempts to discover a solution for TWAVRP will have to solve a VRPTW on numerous occasions. The heuristic proposed in this thesis for solving the TWAVRP builds on and extends some of the methods mentioned above.

4 Outline Heuristic

This section gives a general overview of the heuristic for solving the TWAVRP and the following sections provide more details on each of its components. The heuristic is called the 'Time Window Establisher using Ants for Kernel Enhancing Reiterations' (TWEAKER). It is an Ant Colony Optimization heuristic with three independent colonies that communicate via shared pheromones. One of the three ant colonies creates Time Window Assignments (TWAs) with the goal of finding the optimal Time Window Assignment y^*. To evaluate the quality of a TWA y in terms of travel time it needs to know the routes with minimal expected costs. Finding these routes for y amounts to solving a VRPTW for every demand scenario ω with the time windows as specified by y. Solving these VRPTW problems is the goal of ant colonies number two and three. As in the MACS-VRPTW algorithm (Gambardella et al, 1999), one colony aims for the lowest travel time whereas the other focuses on minimizing the number of vehicles, though the implementation here is quite different from the implementation in MACS-VRPTW. Note that although there are a total of three objectives, TWEAKER is not a multiple objective heuristic as it specifically aims to find y^*, the aim of colony one. As the quality of a TWA y depends on the length of route schedules $r_\omega(y)$ it is natural to have this as an objective for the VRPTW sub-problems. Including vehicle number minimization and preferring it over travel cost minimization may be less obvious as it may not seem directly relevant for finding y^*. This counter-intuitive objective preference greatly improves performance though. Such counter-intuitive preferences are actually not uncommon for Multiple Objective ACO (MOACO) algorithms as the MOACO framework of Angus and Woodward (2003) shows. One reason could be that vehicle number minimization helps escape local minima for travel costs. Barán and Schaerer (2003) modify MACS-VRPTW to solve VRPTW problems while using only one colony, yet they consider the

two objectives to be equally important and search for Pareto optimal solutions. In the present context the only advantage of one colony for the VRPTW would be a reduction in memory usage whereas using two colonies has the advantage that updating pheromones can be done differently for both colonies, which increases system performance (Gambardella et al 1999). Therefore two colonies are used for solving VRPTW problems.
Ant colony number one is called the Time Window Assignment Generator or TWA Generator and ant colonies two and three together form the Route Generator. The TWA Generator and Route Generator share their pheromones τ. TWEAKER initializes the pheromones τ with τ_0 and has the Route Generator solve a VRPTW for each demand scenario ω with time windows equal to the exogenous time windows $[y_v^s, y_v^e], \forall v \in V$. The Route Generator updates τ in the process and gives the updated pheromones τ to the TWA Generator. The TWA Generator uses τ to generate a pool $\mathfrak{J}$ of size $N_{\mathfrak{J}}$ of initial Time Window Assignments. For each TWA $y \in \mathfrak{J}$, TWEAKER begins a separate iteration process. For a y in pool $\mathfrak{J}$, the Time Window Assignment in iteration i is denoted with $y^{(i)}$, with $y^{(0)}$ the original y from pool $\mathfrak{J}$. The pheromones associated with $y^{(i)}$ are denoted as $\tau^{(i)}$. In each iteration i, the TWA Generator creates a new pool $\mathfrak{P}^{(i)}$ of size $N_{\mathfrak{P}}$ of potential new Time Window Assignments. For each TWA $y^{(i)}$ the Route Generator creates route schedules $r_\omega\left(y^{(i)}\right)$ for all demand scenarios ω. For each potential TWA $\mathsf{y} \in \mathfrak{P}^{(i)}$ the Route Generator uses pheromones $\tau^{(i)}$ to generate route schedules. The Time Window Assignment in the next iteration, $y^{(i+1)}$, is the TWA y with the lowest expected costs in pool $\mathfrak{P}^{(i)}$:

$$y^{(i+1)} = \left\{ \mathsf{y} | \mathsf{y} = \arg\min_{\mathsf{y} \in \mathfrak{P}^{(i)}} \mathop{\mathbb{E}}_{\omega \in \Omega} \left[c_{r_\omega(\mathsf{y})} \right] \right\} \tag{2}$$

Once $y^{(i+1)}$ has been chosen from the pool $\mathfrak{P}^{(i)}$, pheromones $\tau^{(i+1)}$ are set equal to the pheromones associated with the y that was chosen to be $y^{(i+1)}$.
The iteration continues until $y^{(i+1)} = y^{(i)}$. This equality means that pool $\mathfrak{P}^{(i)}$ does not contain any candidate with lower expected cost than $y^{(i)}$ and TWEAKER has encountered a local optimum. In such a situation, TWEAKER restores an earlier assignment, $y^{(i+1)} = y^{(i-3)}$, and generates a new pool $\mathfrak{P}^{(i+1)}$. If $i < 3$, $y^{(0)}$ is restored instead. The generation of pools $\mathfrak{P}^{(i)}$ is stochastic and it is likely that $\mathfrak{P}^{(i+1)} \neq \mathfrak{P}^{(i-3)}$. If $\mathfrak{P}^{(i+1)} = \mathfrak{P}^{(i-3)}$ then $\mathfrak{P}^{(i+1)}$ is generated anew until $\mathfrak{P}^{(i+1)} \neq \mathfrak{P}^{(i-3)}$. TWEAKER continues the iteration process with the new pool $\mathfrak{P}^{(i+1)}$. If TWEAKER encounters a local optimum $N_{\mathfrak{T}}$ number of times, the iteration process is terminated. The process is repeated for another initial TWA $y^{(0)} \in \mathfrak{J}$ to which it was not yet applied.
TWEAKER applies this iteration procedure to each $y^{(0)} \in \mathfrak{J}$. Pheromones $\tau^{(0)}$ are the same for all $y^{(0)} \in \mathfrak{J}$. Note that costs associated with $y^{(i+1)}$ do not need to be lower than costs associated with $y^{(i)}$. TWEAKER therefore remembers the TWA $y^{(i)}$ with lowest expected costs across iterations i, denotes with y'. TWEAKER compares y' across pool $\mathfrak{J}$ and returns the one with lowest expected costs as the best Time Window Assignment y^*.
During execution TWEAKER keeps a record of good route schedules. TWEAKER has a list $\mathfrak{L}$ of size $N_{\mathfrak{L}}$ with route schedules with lowest expected costs. $\mathfrak{L}$ is used to quickly check whether TWEAKER already has a good feasible solution for a new TWA $y^{(i)}$. In addition, TWEAKER keeps a list $\mathfrak{B}^{(i)}$ of best feasible route schedules for a TWA $y^{(i)}$. $\mathfrak{B}^{(i)}$ is initialized with route schedules from a nearest neighbour algorithm and if $i > 0$ also with route schedules from $\mathfrak{L}$ and $\mathfrak{B}^{(i-1)}$. $\mathfrak{B}^{(i)}$ is updated during execution of the Route Generator.

The schedule below shows the steps in TWEAKER and the sections that contain a detailed description for a particular step. Note that there is no reference to ant colonies as the three colonies are implemented in the form of the TWA Generator (colony 1) and the Route Generator (colonies 2 and 3).

(0) Initialize Pheromone: Pheromone τ is initialized with τ_0, a pre-specified pheromone struc-

ture.

(1) Create initial TWAs:

(1a) Set the exogenous time windows as the time windows of a VRPTW.

(1b) Let the Route Generator solve this VRPTW and update τ.

(1c) The TWA Generator uses τ to generate a pool $\mathfrak{J}$ of initial Time Window Assignments. Run steps 2 to 3 to completion for each initial TWA $y^{(0)} \in \mathfrak{J}$.

(2) TWA Generation in iteration i:

(2a) Generate a pool $\mathfrak{P}^{(i)}$ of potential TWAs.

(2b) The Route Generator creates for each potential TWA route schedules for each demand scenario.

(2c) $y^{(i+1)}$ is chosen from pool $\mathfrak{P}^{(i)}$ according to equation 2.

(2d) If $y^{(i+1)} = y^{(i)}$, proceed to step (3) else return to step (2a).

(3) Restore an earlier TWA:
An earlier TWA is restored, $y^{(i+1)} = y^{(i-3)}$. If this is done for a threshold number of times, $N_{\mathfrak{T}}$, proceed to step (4), otherwise return to step (2a).

(4) Termination:

(4a) Terminate the current procedure and save the output.

(4b) If $\exists y^{(0)}$ from step 1 that has not been used yet, take that $y^{(0)}$ and return to step 2.

5 Constructing and Updating Pheromone

To find y^*, TWEAKER evaluates numerous Time Window Assignments y. To evaluate one TWA amounts to solving a VRPTW for each demand scenario. The number of VRPTW problems to solve therefore depends on $|\Omega|$, $N_{\mathfrak{P}}$ and the duration of the $N_{\mathfrak{J}}$ iteration processes. The pheromones in ACO algorithms help to solve this number of VRPTWs in a reasonable amount of time, because once one VRPTW is solved by an ACO algorithm the information needed to solve it is stored in pheromone τ and can be reused to create solutions for other VRPTWs. MACS-VRPTW (Gambardella et al 1999) is able to solve a VRPTW quickly but it constructs pheromones as lying on an arc between two customers. It does not capture time information, as it may be much less desirable to visit customer w after v at time t than it is at time t' due to the time window constraints. A TWA y may have quite different endogenous time windows for each customer than another TWA y' may have and the optimal route schedule for y, $r^*(y)$, does not need to have any segment in common with $r^*(y')$. Therefore the pheromone matrix that MACS-VRPTW constructs to solve the VRPTW for y may not lead to good solutions for Time Window Assignment y'.
Rather than having pheromones on the arcs between customers, TWEAKER constructs pheromones as lying across the exogenous time windows of the customers. Construction of pheromone in this manner confers certain strengths to TWEAKER: the pheromone level is defined over the entire exogenous time window of a customer v and each TWA y defines for every customer v an endogenous time window within the exogenous one. The average pheromone levels across the time windows in a Time Window Assignment y can therefore be used as an a priori indication of the quality of y. This is useful for constructing new TWAs.
As time is continuous, pheromone levels must be determined by means of a function that can return an appropriate value for each moment in the exogenous time window. The approach employed here was inspired by the idea of Gaussian kernels mentioned in Blum (2005) and Socha

and Dorigo (2008) and works as follows. For each customer v, define a set K_v of kernels. Each kernel defines a pheromone level for a part of the exogenous time window. This happens in layers: the first kernel covers the entire exogenous time window and defines a pheromone level for it, then the exogenous time window is split in two equal parts, each with its own kernel with equal area, then a third layer breaks up the exogenous time window in four parts and this continues until the last layers, yielding a total of $2^l - 1$ kernels, with l the number of layers. Figure 1 shows four layers of kernels along the exogenous time window on the horizontal axis. In figure 1 the vertical axis is the pheromone level. Summed together, the kernels define one pheromone level for the entire time span, which would be the black line in figure 1.

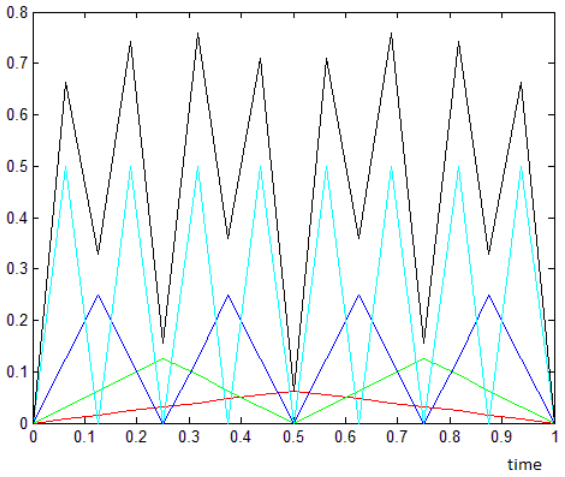

Figure 1: Triangular Kernels

As can be seen from figure 1, a kernel $k \in K_v$ has three points that define a triangular shape. These three are a top x_k^t and two base points x_k^l and x_k^r. For each moment within the time span covered by x_k^l and x_k^r kernel k defines a positive pheromone level. Outside this span the pheromone level of kernel k equals zero. The bell curves of the Gaussian kernels are replaced with a triangular shape because such shapes are computationally much more efficient and there is no real disadvantage. A kernel k also has two minima and two maxima that limit the area where top x_k^t can be situated. Along the exogenous time window, top x_k^t is restricted by borders b_k^l and b_k^r, $b_k^l \leq x_k^t \leq b_k^r$, and vertically it is constrained by a minimum and maximum height, $h_{min} \leq x_k^t \leq h_{max}$. The surface below a kernel k is held constant at a value z_k and as the top increases or decreases, the base points come nearer or move farther from each other. The base points of a kernel k have an equal distance to the top of k to maintain symmetry, yet one of them may bump into the edge of the exogenous time window. If that happens, the base point that stands on the edge comes to lie at the place where the intersection between the kernel and the edge would otherwise have been. The other base point is moved such that the area below the kernel stays equal to z_k. Figures 2 and 3 show how top and base points are limited.
The fixed surface z_k together with the position of x_k^t defines the base points as:

$$x_{k,1}^l = x_{k,1}^t - \frac{z_k}{x_{k,2}^t}, \quad x_{k,2}^l = 0 \tag{3}$$

$$x_{k,1}^r = x_{k,1}^t + \frac{z_k}{x_{k,2}^t}, \quad x_{k,2}^r = 0 \tag{4}$$

Where $x_{k,1}^t$ denotes the horizontal position of the top, $x_{k,2}^t$ its vertical position and similarly for the base points. The formulas illustrate that computations with triangular kernels are straightforward and hence will be fast. Pheromone $\tau_k(t)$ from kernel k at time t is calculated as:

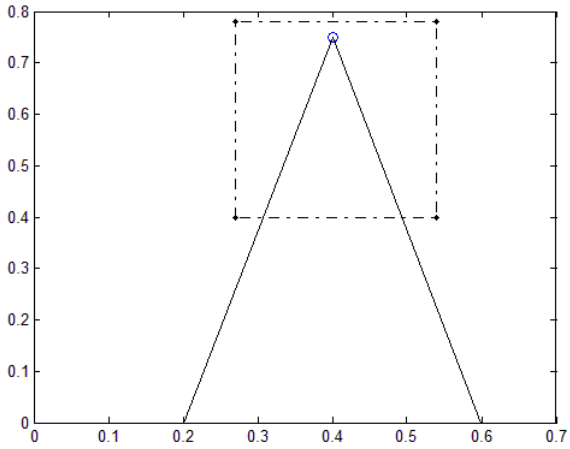

Figure 2: Boundaries to the kernel top

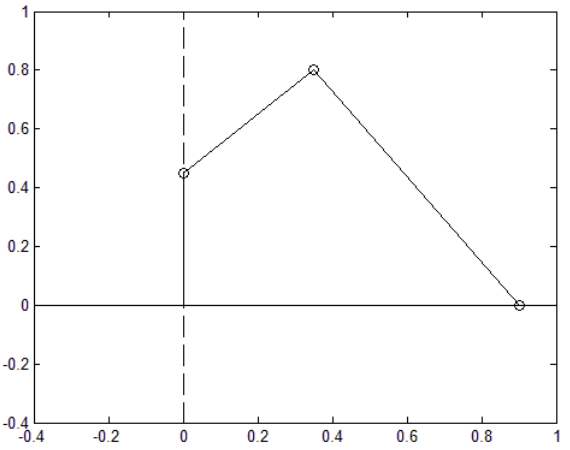

Figure 3: The base point at the edge of the exogenous time window

$$\tau_k(t) = \begin{cases} 0 & \text{if } t < x^l_{k,1} \\ \dfrac{t - x^l_{k,1}}{x^t_{k,1} - x^l_{k,1}} x^t_{k,2} & \text{if } x^l_{k,1} \leq t \leq x^t_{k,1} \\ \dfrac{x^r_{k,1} - t}{x^r_{k,1} - x^t_{k,1}} x^t_{k,2} & \text{if } x^t_{k,1} \leq t \leq x^r_{k,1} \\ 0 & \text{if } t > x^r_{k,1} \end{cases}$$

In other words, the pheromone level at a time t for a kernel k is halfway between the top and the base point between which the time t happened to fall. Pheromone for customer v at time t is calculated as:

$$\tau_v(t) = \sum_{k \in K_v} \tau_k(t) \tag{5}$$

By using triangular kernels, pheromone can be continuous rather than discrete and updating will be done by adjusting the position of the top. The position of the base points will then be calculated as in equations 3 and 4. A general issue that may arise in updating pheromone is that remote customers may at any time have low pheromone and therefore may keep being picked last by ants constructing routes. These remote customers then have to be included at an unfavorable moment, which deteriorates costs. An advantage of keeping the surface z_k per kernel k constant is that pheromone levels will on average be equal across customers. Through this normalisation of pheromones the average desirability of visiting customers is equal and there is a better chance of including a remote customer when an ant visits another customer that is relatively nearby.
$K_v, v \in V \backslash \{0\}$ will be referred to as a 'Kernel Template' from here on. Each kernel is updated separately. Pheromone is updated to include new information on the new route schedules that the ants generated. The evaporation parameter ρ lowers pheromone and the parameter θ increases pheromone, $\rho < 1$ and $\theta > 1$. Evaporation allows ants to forget previous solutions and is applied every time an ant visits a customer, so that other ants are less likely to generate similar routes. Increasing pheromone is done with the best route schedules found so far and biases the search to the neighbourhood of these best solutions:

$$x^t_{k,2} \leftarrow \rho x^t_{k,2}, \qquad \text{if } b^l_k \leq t^{v,\omega}_r \leq b^r_k \tag{6}$$

$$x^t_{k,2} \leftarrow \theta x^t_{k,2}, \qquad \text{if } b^l_k \leq t^{v,\omega}_{r^*} \leq b^r_k \tag{7}$$

$$x^t_{k,1} \leftarrow \sum_{\omega, r^*} \left(t^{v,\omega}_{r^*} - x^t_{k,1} \right) \qquad \text{if } b^l_k \leq t^{v,\omega}_{r^*} \leq b^r_k \tag{8}$$

Here r^* is a route schedule from the population of best route schedules $\mathfrak{L}$. Newly generated route schedules lower the tops of the kernels for which their arrival time $t^{v,\omega}_r$ falls within the borders b^l_k and b^r_k (equation 6). The tops of the kernels are increased if the arrival time of a route schedule r^* falls within the borders (equation 7). The tops are horizontally moved to the average arrival times within their borders of the best route schedules as is shown by equation 8. After the top is updated, the base points x^l_k and x^r_k are updated according to equations 3 and 4.
The list $\mathfrak{L}$ is initialized with route schedules from a nearest neighbour heuristic. Pheromones τ are thus a collection of kernel templates, $\tau = \{K_v\}_{v \in V}$.

6 Generating Routes

The Route Generator receives an assignment y and pheromones τ as input and it creates route schedules for each demand scenario ω. To do so, there are two colonies and for each colony there are $|\Omega| \cdot m$ number of ants, with m ants for each demand scenario ω. In one run of the Route Generator, each ant visits all customers, starting and ending at the depot and visiting the depot in between *only* if so forced by time window and capacity constraints. If an ant visits the depot, his capacity becomes Q and if he can still visit customers within the time window constraints he will continue to visit customers. If he cannot go to any unvisited customers within the time windows constraints, his time is reset to the opening time of the depot. Each route along customers that an ant travels without resetting its time corresponds to a route that a vehicle from the depot would travel. Each ant thus creates a route schedule r_ω for the demand scenario ω it is associated with. Each time an ant visits a customer, pheromone at that customer is evaporated. After each entire run, pheromone at a customer v is increased at the arrival times of the best route schedules at customer v. There are $N_{\mathfrak{R}}$ runs resulting in $N_{\mathfrak{R}}$ route schedules per scenario. As mentioned there is a population $\mathfrak{B}$ of size $N_{\mathfrak{B}}$ of best route schedules. If a newly generated route schedule has a lower number of vehicles than the one in $\mathfrak{B}$ with the highest number of vehicles, it replaces that route schedule. If all route schedules in $\mathfrak{B}$ have the same number of vehicles or less as the new route schedule, but the new schedule has lower travel costs than one in $\mathfrak{B}$ with an equal number of vehicles, then the new route schedule replaces the one with higher travel costs. A route schedule r with a lower number of vehicles than a route schedule r' is preferred over r' even if r' has lower travel costs. As in MACS-VRPTW, this preference ordering is necessary to construct good schedules fast.

There are two separate ant colonies for solving a VRPTW. One aims for travel cost minimization, the other for minimization of the number of vehicles. They keep their kernel templates separate but share the population of best solutions. The two colonies work the same except that for the colony minimizing travel costs, a local search is applied to a route schedule once an ant is done constructing the schedule. This local search is the CROSS exchange as discussed in Taillard et al (1997). The CROSS exchange generalizes local searches such as 2-opt and Or-opt by allowing whole segments of routes to be swapped.

One run of the Route Generator works as follows: at the start, each ant visits a randomly chosen customer. Ants walk sequentially rather than parallel, so an ant a does not encounter dynamically changing pheromone. Sequentially walking ants make the implementation simpler because pheromone is defined along the exogenous time window of customers. This time dimension would make an algorithm with parallel walking ants more complicated and slow as it would need to keep count of who arrives where next. Each time an ant a visits a customer v it retrieves an amount d_v^ω, the demand of customer v at demand scenario ω, and its capacity decreases. It updates the remaining capacity and time and removes that customer from his list of customers to visit, Λ_a. Initially $\Lambda_a = V \backslash \{0\}$, the set of customers. Due to time window and capacity constraints the ant may not be able to choose his next destination among all unvisited customers. Let $\widetilde{\Lambda_a} \subset \Lambda_a$ denote the unvisited customers that meet the time window and capacity constraints and can therefore serve as next destinations. If the ant returns to the depot and both time and capacity are restored, the list is restored as well: $\widetilde{\Lambda_a} = \Lambda_a$.

The list $\widetilde{\Lambda_a}$ may still be quite large. Blum (2005) suggests to have the ant choose only among the $N_{\mathfrak{N}}$ neighbours that are closest in terms of travel time. Let the list of $N_{\mathfrak{N}}$ nearest neighbours be denoted by $\widetilde{\Lambda_a^b} \subset \widetilde{\Lambda_a}$. The next destination that could be part of the optimal solution may not be contained in $\widetilde{\Lambda_a^b}$ but it seems reasonable that the ant can find a good route by looking at its $N_{\mathfrak{N}}$ nearest neighbours, as long as $N_{\mathfrak{N}}$ is sufficiently large to allow construction of schedules with low routes and travel costs. If the aim is to find good solutions quickly, $\widetilde{\Lambda_a^b}$ is used, if the best routes possible need to be found $\widetilde{\Lambda_a}$ is used instead. The probability with which ant a chooses a next destination w when he is at v is given by:

$$p^a_{v,w}(t) = \frac{\tau^\alpha_w(t')\eta^\beta_{v,w}\nu^{\gamma_1}_{w,1}(t')\nu^{\gamma_2}_{w,2}(t')\phi^\delta_{v,w}}{\sum_{w' \in \widetilde{\Lambda^b_a}} \tau^\alpha_{w'}(t')\eta^\beta_{v,w'}\nu^{\gamma_1}_{w,1}(t')\nu^{\gamma_2}_{w,2}(t')\phi^\delta_{v,w'}}, \quad w \in \widetilde{\Lambda^b_a} \tag{9}$$

Here t is the current time, v is the current location of the ant, w is a potential next location, t' is the arrival time at w, α, β, γ_1, γ_2 and δ are parameters to be explained later on, τ denotes pheromone, η denotes visibility, ν denotes urgency and ϕ is frequency. Visibility $\eta_{v,w}$ is one divided by the distance between v and w. Urgency comes in two forms, mirroring its appearance in MACS-VRPTW: $\nu_{w,1}$ measures how much time remains before the time window of w closes. The closer t' is to the closing time of the time window of w the larger the urgency will be. $\nu_{w,2}$ measures how much time passes between ending of service at v and the start of service at w. Frequency $\phi_{v,w}$ measures how often ants previously choose location w after location v. In other words, the probability that an ant chooses a location w if it is at location v depends on the pheromone level at w at the time when it arrives at w, on the distance between v and w, on how much time remains before w cannot be visited any more, on how much time passes between services and finally on how often w has been picked after v by other ants. α, β, γ_1, γ_2 and δ regulate how important these five factors are relative to each other.

The frequency factor is mainly included to escape stagnation and δ equals zero until such a situation occurs. The Route Generator does not and cannot prevent the ants from creating the same routes over and over again. It is theoretically possible that if the ants create a thousand routes, only ten of them are unique. It is not a problem that ants may occasionally create the same routes as long as the better routes are found. However, if the solution no longer improves, a situation of stagnation, then it does become a hindrance. The frequency factor lowers the probability of w being chosen after v if it was chosen after v often in the past and therefore lowers the probability of creating the same routes repeatedly. Frequency is not necessary to find good solutions to a VRPTW problem but increases the speed with which good solutions are found and helps to maintain diversity. The size of $\widetilde{\Lambda^b_a}$ is increased if stagnation sets in and more diversity is desired.

ACO algorithms also incorporate a probabilistic rule to balance exploration and exploitation (Dorigo and Stützle, 2004). Given the probabilities $p^a_{v,w}(t)$, there is a probability q_0 that customer w is chosen for which $p^a_{v,w}(t)$ is the largest and there is a probability $1-q_0$ that the choice for a next location w will happen with a chance proportional to probability $p^a_{v,w}(t)$. The former would be exploitation and the latter exploration. Exploitation adds a greedy element and is meant to create routes with short travel times, whereas exploration is meant to maintain search diversity and to allow for new, unexplored routes to be discovered.

7 Generating Assignments

This sections explains how the Time Window Assignment Generator assembles new Time Window Assignments y. The TWA Generator takes a Time Window Assignment y and pheromones τ as input and generates a pool of candidate TWAs. There are two different ways to do so, one meant for the initialization process and the other for the iteration process. At the start, TWEAKER creates a pool $\mathfrak{J}$ of TWAs. Rather than choosing one TWA y from $\mathfrak{J}$, each Time Window Assignment y in $\mathfrak{J}$ is explored to ensure diversity and search a wider area of the solution space. TWEAKER attempts to improve each TWA $y \in \mathfrak{J}$ with an iteration process. At iteration i, the TWA Generator takes $y^{(i)}$ and pheromone $\tau^{(i)}$ as input and creates a pool $\mathfrak{P}^{(i)}$ of candidate TWAs, from which one is chosen to be $y^{(i+1)}$ according to equation 2. A Time Window Assignment in a pool such as $\mathfrak{J}$ and $\mathfrak{P}^{(i)}$ is assembled by first specifying for each customer $v \in V \setminus \{0\}$ a number of candidate endogenous time windows and subsequently letting

an ant choose one time window for each of these customers. This ant comes from an ant colony that is independent from the two colonies used in the Route Generator.

7.1 Candidate Time Windows

The ACO algorithm of the previous section needs candidate time windows for every customer. For pool $\mathfrak{J}$ these are created by considering the solutions of the VRPTW problems with exogenous time windows. There is one VRPTW for each demand scenario ω, so for every customer v the solutions provide $|\Omega|$ arrival times. These are at the centre of the $|\Omega|$ candidate time windows that the TWA Generator uses to create pool $\mathfrak{J}$.
The candidate time windows that the TWA Generator needs for a pool $\mathfrak{P}$ are created with help of time factors m. The TWA Generator is given a TWA $y^{(i)}$. For each customer v, the time window of customer v in $y^{(i)}$ can be moved to the past or to the future, so there are two time factors per customer, a time factor m_v^- for the past and a time factor m_v^+ for the future. Each is restricted: $0 < m_v^- < 1$ and $0 < m_v^+ < 1$. These turn the time window of $y^{(i)}$ at v into two candidates, $[y_v - m_v^- w_v, y_v + w_v - m_v^- w_v]$, denoted with c_v^-, and $[y_v + m_v^+ w_v, y_v + w_v + m_v^+ w_v]$, denoted c_v^+, which together with the time window $[y_v, y_v + w_v]$ from $y^{(i)}$, denoted c_v, form three time window candidates for each customer v.
The time factors indicate how much a time window is moved to the past or future to create new candidates. The time factors are asymmetric, so c_v^- can be much further into the past than c_v^+ lies in the future and time factors across customers can be very different. They can be quite small for one customer, implying that its time window is approximately at the place it should be, whereas for another customer the time factors can still be quite large, implying the time window is not yet in its rightful place. After the ants have used these candidate time windows to generate a pool $\mathfrak{P}^{(i)}$ of candidate TWAs and a new TWA $y^{(i+1)}$ has been chosen from $\mathfrak{P}^{(i)}$, the time factors are updated based on which candidate time windows constitute $y^{(i+1)}$. If the indicator function $\mathbb{1}_{\{c_v,h\}}$ equals one if c_v in iteration h was chosen in iteration h to be the time window for $y^{(h+1)}$ and zero otherwise, then the formula for updating the time factors is:

$$m_{v,i+1}^+ = \sum_{p=0}^{P} \mathbb{1}_{\{c_v^+,i-p\}} w_p m_{v,i-p}^+ \tag{10}$$

$$m_{v,i+1}^- = \sum_{p=0}^{P} \mathbb{1}_{\{c_v^-,i-p\}} w_p m_{v,i-p}^- \tag{11}$$

The time factors in iteration $i+1$ depend on the time factors of the previous P iterations, which are weighted with w. P is chosen to be two and $w = \{2, 1, 0.5\}$. If a time window is continually moved to the past then it should do so faster each time, whereas if a time window moves back and forth in past and future then it is somewhere near the place it should settle and its time factors should decrease.
The time factors are initialized as $m_{v,0}^+ = m_{v,-1}^+ = m_{v,-2}^+ = 0.5$ and similarly for m_v^-. If after updating the time factors for all customers fall below a threshold $T_{\mathfrak{M}} = 0.1$, then the new TWA is too similar to its predecessor and the process is considered to have arrived at a local optimum. TWEAKER then sets $y^{(i-3)}$ as $y^{(i+1)}$ and resets the time factors to their initial value. TWEAKER retraces its steps three iterations and tries again. If this resetting has happened $N_{\mathfrak{T}}$ number of times, TWEAKER terminates the current iteration process.

7.2 TWA Generator ACO

Suppose that for each customer the candidate time windows have been determined. An ant a assemble a TWA from these candidates by selecting a new destination from a list Λ of the $N_{\mathfrak{N}}$

nearest customers. If ant a is at a candidate time window h of customer v it chooses to go to a candidate time window l of customer w with probability:

$$p^a_{v_h,w_l} = \frac{\overline{\tau}^{\alpha}_{w_l} \eta^{\beta}_{v_h,w_l}}{\sum_{w'_{l'} \in \Lambda_h} \overline{\tau}^{\alpha}_{w'_{l'}} \eta^{\beta}_{v_h,w'_{l'}}}, \qquad w_l \in \Lambda_h \tag{12}$$

In this formula $\overline{\tau}_{w_l}$ is the average pheromone level over the time span of candidate time window l at customer w. η_{v_h,w_l} is the visibility, which is given by:

$$\eta_{v_h,w_l} = \frac{\max\left(\min\left((y^e_h + t_{v,w}), y^e_h\right) - \max\left((y^s_h + t_{v,w}), y^s_l\right), 0\right)}{y^e_l - y^s_l} \tag{13}$$

Here $t_{v,w}$ is the travel time between v and w. For a time window h, y^s_h is the start of time window h and y^e_h is the end of time window h. Visibility raises the probability that ant a chooses a candidate time window that starts soon after the time window it is currently located, correcting for travel time $t_{v,w}$. This ensures that the time windows stay coupled, which makes it easier to generate routes. Each ant a starts at the depot and visits customers until a candidate time window for each customer has been chosen. Each ant a keeps track of time nor capacity and does not visit the depot in between visiting customers. The aim of the ants is not to generate route schedules but to choose those time windows that connect best to enable the Route Generator to find good routes. Once a TWA is chosen, its population of best route schedules must be initialized. TWEAKER does this by checking whether route schedules from a previous population and from the list of best route schedules $\mathfrak{L}$ are feasible. TWEAKER also uses a nearest neighbour heuristic to create route schedules for the new TWA. The value of the newly created TWA are the travel costs of the best route schedules that the Route Generator can find for it.

8 Results

This section shows the numerical experiments and results. All computations in this section were carried out on a Windows 7 system with an Intel Core i5 2.4 GHz processor and 8 GB RAM. All code was written in Java. The results compare TWEAKER's solutions to two benchmarks. The first benchmark consists of modified versions of Solomons benchmark solutions and the second benchmark are the TWAVRP instances for which Spliet and Gabor (2012) have calculated the optimal solutions.

8.1 Solomon Instances

Solomon (1987) created six sets of problems, called R1, C1, RC1, R2, C2 and RC2, which are considered the standard benchmark for VRPTW problems. Each set contains between eight and twelve instances and the six sets together contain 56 instances. For a particular problem set, the customer locations, service time, demand and vehicle capacity are the same across instances in that set but time windows differ, both in window width and start time. All sets contain 100 customers. In problem sets R1 and R2 the customer locations are randomly generated, for C1 and C2 the locations are clustered and for RC1 and RC2 the customer locations are mixed, some randomly generated and some clustered. Problem sets R1, C1 and RC1 have small vehicle capacity combined with tight time windows, whereas problem sets R2, C2 and RC2 have large vehicle capacity and wide time windows. Solutions to instances in sets R2, C2 and RC2 therefore tend to have fewer routes and more customers per route than solutions for instances in sets R1, C1 and RC1. Sets C1 and C2 have time windows whose placement is based on a solution to a similar VRP problem without time windows.

These problem instances are modified into TWAVRP problems by generating the set of demand scenarios Ω and creating exogenous time windows from the problem sets themselves. A set Ω of 30 demand scenarios is generated for each problem set by applying a basic bootstrap (Dolnicar and Leisch, 2009) to the demand of the corresponding problem set and all demand scenarios have equal occurrence probability. Solomon (1987) provided each customer in each instance of a problem set with a time window. For the moment, these are referred to as the endogenous time windows. Since a customer is the same across instances of a problem set except for the endogenous time windows, it becomes possible to create exogenous time windows. The exogenous time window for a customer in a problem set is such that the endogenous time windows in the instances are always within the exogenous time window. Thus the exogenous time window for a customer in a problem set is the same across instances. The exogenous time window of a customer does not start earlier than the opening time of the depot plus the travel time from the depot to that customer and similarly for the end of the exogenous time window. For a customer v in a problem set A, let y_v^p be earliest moment that a time window starts across the instances in A and y_v^g be the latest moment. $t_{0,v}$ is the travel time from the depot to v and $t_{v,0}$ the travel time from v to the depot, which differs from $t_{0,v}$ due to service time. The exogenous time window $[y_v^s, y_v^e]$ of customer v in problem set A is then given by:

$$y_v^s = \max\left(y_v^p, y_0^s + t_{0,v}\right)$$
$$y_v^e = \min\left(y_v^g, y_0^e - t_{v,0}\right)$$

Each instance in a problem set can be regarded as one assignment of endogenous time windows for which a VRPTW problem needs to be solved.

To see whether the Route Generator is capable of solving the original Solomon instances, tables 1 and 2 show the optimal solutions for the original Solomon instances and the solutions of the Route Generator (RG). The results in tables 1 and 2 were obtained with parameters $\alpha = 0.5$, $\beta = 2$, $\gamma_1 = 0.5$, $\gamma_2 = 2$ and $\delta = 0.5$. These values were found by running a grid search, where each parameter could take the values 0.5, 1, 2 and 5. The obtained values give the lowest expected costs when applied to the Solomon instances. Experimentation with setting one or more of the factors equal to zero did not lead to lower expected costs. Note that here $\gamma_1 \neq \gamma_2$, so the factors regulating the two components of urgency are not equal. This shows that it may have been beneficial to break urgency into two components instead of keeping it as one as in MACS-VRPTW. The factor α in equation 9 indicates the importance of pheromone with regard to the other factors, visibility, urgency and frequency. In MACS-VRPTW α is set to 1 but here it was found that $\alpha = 0.5$ gives lower expected costs, only slightly lower compared to setting $\alpha = 1$, keeping the other factors equal. The number of ants was set to 10 as in MACS-VRPTW. The pheromone parameters, ρ for evaporation and θ for updating, were set to $\rho = 0.9$, as is standard in the literature, and $\theta = \frac{1}{\rho}$. Experimentation indicates that setting the number of layers to five and surface z_k per kernel to 0.25 times the length of the exogenous time window yields pheromone values with a good balance between discovering new route schedules (exploration) and finding route schedules with lowest expected costs (exploitation).

In tables 1 and 2 the columns with either 'R1', 'C1', 'RC1', 'R2', 'C2' or 'RC2' at the top represent the optimal solutions for the Solomon instances of the corresponding problem set, the columns with 'RG' at the top represent the solution according to the Route Generator and the columns with 'Time(s)' indicate the time in seconds that the Route Generator took to come to the solution. TWEAKER will be tested only on the first 25 customers of the instances, therefore the solutions in tables 1 and 2 concern only the first 25 customers.

These tables show that the Route Generator can find solutions within 10 per cent from the optimal solution, except for problem sets R2 and C2 where the solutions from the Route Generator are 16 per cent from the optimal solutions. As R2, C2 and RC2 are the problem sets with the wider time windows and larger vehicle capacity, their space of feasible solutions is larger. This

Instance	R1	RG	Time(s)	C1	RG	Time(s)	RC1	RG	Time(s)
1	617.10	635.76	45.40	191.30	191.83	40.86	461.10	465.65	37.68
2	547.10	604.71	45.99	190.30	195.38	44.84	351.80	355.09	43.89
3	454.60	488.18	49.72	190.30	197.00	53.43	332.80	340.26	48.04
4	416.90	443.47	52.52	186.90	194.65	56.73	306.60	319.69	50.69
5	530.50	553.58	45.41	191.30	191.83	42.31	411.30	420.00	41.52
6	465.40	494.77	47.56	191.30	191.83	40.96	345.50	346.93	39.27
7	424.30	451.23	50.98	191.30	191.83	41.78	298.30	299.52	43.55
8	397.30	426.81	52.46	191.30	191.66	44.33	294.50	301.28	47.70
9	441.30	462.83	44.89	191.30	191.83	46.39			
10	444.10	454.8	47.64						
11	428.80	463.93	49.85						
12	393.00	412	50.99						

Table 1: Route Generator performance for R1, C1 and RC1

Instance	R2	RG	Time(s)	C2	RG	Time(s)	RC2	RG	Time(s)
1	463.30	502.53	47.41	214.70	247.29	46.11	360.20	387.94	49.14
2	410.50	474.14	51.40	214.70	247.75	52.86	338.00	369.66	55.96
3	391.40	428.39	58.30	214.70	247.77	60.51	326.90	345.19	62.26
4	355.00	421.25	61.46	213.10	245.52	65.06	299.70	317.57	67.15
5	393.00	443.86	50.45	214.70	240.6	45.87	338.00	350.85	53.70
6	374.40	438.86	54.23	214.70	258.13	46.51	324.00	375.8	54.07
7	361.60	431.64	61.24	214.50	256.68	46.68	298.30	331.69	56.56
8	328.20	405.55	64.87	214.50	252.73	45.82	269.10	313.88	67.83
9	370.70	415.06	52.80						
10	404.60	460.71	55.25						
11	350.90	379.46	58.63						

Table 2: Route Generator performance for R2, C2 and RC2

makes them considerably harder to solve than problem sets R1, C1 and RC1, which is reflected in the performance of the Route Generator. It takes on average 51 seconds to execute these results. The average deviation from the optimal solution is 8.28 per cent and if the number of runs is lowered such that the Route Generator only needs on average 25 seconds, then the average deviation becomes 17.8 per cent. However, TWEAKER needs to consider a rather large number of VRPTW problems, one for each potential new TWA in pool $\mathfrak{P}(\mathfrak{i})$ for all iterations in all iteration processes. Therefore the number of runs $N_{\mathfrak{R}}$ was lowered so that the Route Generator only needs two to three seconds to solve a VRPTW.

Besides the parameters mentioned earlier, TWEAKER has parameters $N_{\mathfrak{B}}$, $N_{\mathfrak{L}}$, $N_{\mathfrak{N}}$, $N_{\mathfrak{J}}$, $N_{\mathfrak{P}}$ and $N_{\mathfrak{T}}$ that can be optimized after the previous ones have been determined. $N_{\mathfrak{B}}$ controls the size of population $\mathfrak{B}$ and $N_{\mathfrak{L}}$ that of list $\mathfrak{L}$. Large $N_{\mathfrak{B}}$ and $N_{\mathfrak{L}}$ make it more likely that the Route Generator starts with good routes which can guide it to better solutions faster. Yet if $N_{\mathfrak{B}}$ and $N_{\mathfrak{L}}$ are too large, too many solutions are saved which slows the algorithm without providing a justifiable gain in solution quality. Experiments showed that a balance is struck if both are set equal to the number of customers. $N_{\mathfrak{N}}$ controls the size of $\widetilde{\Lambda_a^b}$, the list of nearest neighbours from which ants choose their next destination. Keeping this list small narrows the choice to a few nearby customers and affects both computation time and solution quality positively. However, it may also hinder building diverse solution and prevent the Route Generator from finding the best solutions. Some experimentation suggests that for instances with 50 and 100 customers setting $N_{\mathfrak{N}}$ to a value between 5 to 10 has a positive impact on the route schedules created but for 25 customer instances changing $N_{\mathfrak{N}}$ had little to no effect.

The three parameters $N_{\mathfrak{J}}$, $N_{\mathfrak{P}}$ and $N_{\mathfrak{T}}$ effectively control how often TWEAKER creates new TWAs and solves new VRPTWs and they determine to a large extent the computation time. $N_{\mathfrak{J}}$ is the size of pool $\mathfrak{J}$ and determines how many initial TWAs are formed from the candidate time windows across the entire exogenous time window of each customer. These initial TWAs can be very different and cover completely different areas of the exogenous time windows, therefore a large $N_{\mathfrak{J}}$ maintains diversity in TWA generation. $N_{\mathfrak{P}}$ is the size of a pool $\mathfrak{P}$. It determines the number of candidate TWAs from which TWEAKER can choose the next TWA. If $N_{\mathfrak{P}}$ becomes larger then it will be more likely that TWEAKER finds a better candidate TWA, so it stalls the stagnation of TWA improvement. A large $N_{\mathfrak{P}}$ also implies that the neighbourhood of candidate TWAs is searched more intensively because more candidates are evaluated. In other words, $N_{\mathfrak{J}}$ influences the diversity of the search whereas $N_{\mathfrak{P}}$ is important for the intensity. To maintain reasonable computation times, a balance must be struck between the two.

Figures 4 and 5 show how changes in $N_{\mathfrak{J}}$ (figure 4) and $N_{\mathfrak{P}}$ (figure 5) have an impact on the solution quality for the representative problem instances R101, C101, RC101, R201, C201 and RC201. The horizontal axis depicts the pool size and the vertical axis shows TWEAKERs solution divided by the optimal solution for these Solomon instances. Unfortunately for all six problem instances, there is no obvious trend in these series or a slight decrease at best. In the case of $\mathfrak{J}$ this means that the number of initial TWAs cover too little of the solution space and in the case of $\mathfrak{P}$ this means that additional intensification does not help to discover a better local optimum.

Tables 3 and 4 show the results for solving the modified Solomon instances. For each instance the window width w_v for customer $v \in V\backslash\{0\}$ equals the window width in the original problem instance. The number of customers considered is 25 and the number of demand scenarios is 30. As can be seen from the tables, TWEAKER was run approximately 10 minutes per instance. Basically, the original Solomon instances constitute a Time Window Assignment and TWEAKER has tried to find one with lower expected costs across demand scenarios in Ω. The original Solomon time windows were feasible for the demand scenarios as created with the bootstrap. For some instances TWEAKER succeeded to find a TWA with lower expected costs, especially for the instances of set R1 and RC1. R1 is the set where locations are randomly generated and where there are short scheduling horizons and RC1 is the set where some locations are randomly

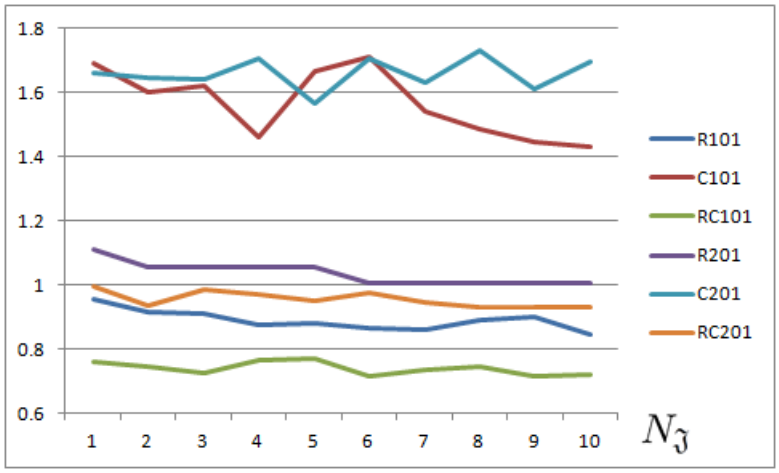

Figure 4: Solution as percentage of optimal solution for varying size of N_3

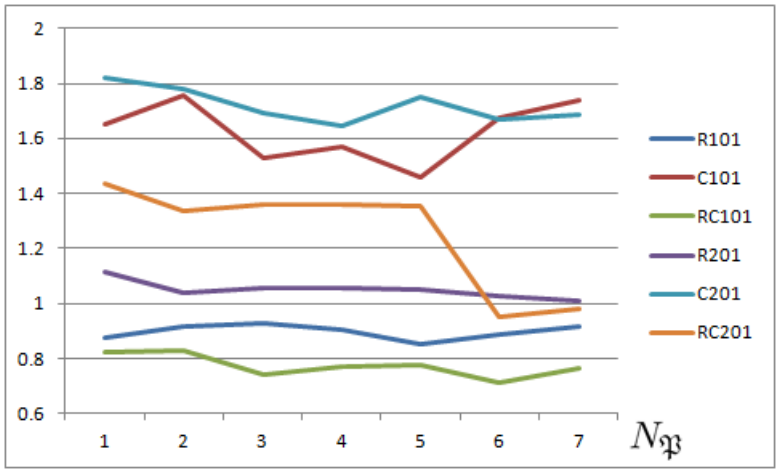

Figure 5: Solution as percentage of optimal solution for varying size of $N_{\mathfrak{P}}$

generated and some clustered and where scheduling horizons are also short. For R1 the average improvement was 2.69 per cent and for RC1 it was 8.43 per cent. There are large differences between instances in whether TWEAKER succeeded to improve upon the expected costs of Solomons original assignment or not. In particular, TWEAKER failed to find a TWA with better expected costs for instances from sets C1 and C2, the purely clustered problems. This can be explained by the fact that Solomon based the placement of his time windows on the solution of the problem without time windows. This increases the difficulty of finding a better TWA within the time limit of ten minutes. It may also be that for clustered problems it is simply harder to find good Time Window Assignments because the optimal assignment of a time window to a customer may be very sensitive to the assignments of its close neighbours. That would be a question for further research.

Instance	R1	RG	Time(s)	C1	RG	Time(s)	RC1	RG	Time(s)
1	617.10	544.78	565.26	191.30	319.25	599.29	461.10	354.58	529.41
2	547.10	558.19	591.31	190.30	286.75	639.26	351.80	319.06	588.07
3	454.60	474.61	659.84	190.30	216.04	755.03	332.80	308.79	679.82
4	416.90	425.90	705.35	186.90	202.01	814.43	306.60	296.53	731.11
5	530.50	466.53	575.98	191.30	237.63	614.16	411.30	345.26	564.58
6	465.40	438.03	636.59	191.30	304.05	607.26	345.50	312.91	586.10
7	424.30	415.02	692.43	191.30	226.00	666.48	298.30	300.69	656.62
8	397.30	404.22	733.33	191.30	217.53	669.45	294.50	295.01	721.85
9	441.30	442.90	622.91	191.30	216.83	700.77			
10	444.10	414.70	680.89						
11	428.80	426.52	683.19						
12	393.00	362.74	737.96						

Table 3: TWEAKER performance for R1, C1 and RC1

Instance	R2	RG	Time(s)	C2	RG	Time(s)	RC2	RG	Time(s)
1	463.30	489.42	658.14	214.70	336.40	677.44	360.20	342.96	647.16
2	410.50	424.01	684.56	214.70	300.58	692.13	338.00	347.64	709.13
3	391.40	395.67	798.69	214.70	252.78	838.88	326.90	310.80	813.05
4	355.00	380.15	860.05	213.10	253.70	889.26	299.70	301.67	859.63
5	393.00	401.02	706.44	214.70	295.07	637.47	338.00	345.54	700.59
6	374.40	400.02	760.53	214.70	290.09	685.63	324.00	320.30	702.58
7	361.60	364.34	842.74	214.50	279.78	695.84	298.30	313.86	755.82
8	328.20	365.06	876.36	214.50	275.41	711.82	269.10	300.80	827.70
9	370.70	389.66	743.88						
10	404.60	411.30	755.49						
11	350.90	342.27	803.17						

Table 4: TWEAKER performance for R2, C2 and RC2

8.2 Optimally Solved TWAVRP Instances

Spliet and Gabor (2012) have calculated the optimal solutions to five TWAVRP instances, each with eight customers and three demand scenarios. As described in their paper, customers are randomly generated and the generation of demand and time windows is based on their experience with the distribution centre of a Dutch retailer. For these five instances TWEAKER manages

to find near optimal solutions as shown in table 5 below. The parameters used were the same as the ones for the Solomon instances above.

Instance	Optimal Solution	Time(s)	TWEAKER	Time(s)
1	14.92	547.58	14.92	4.80
2	17.31	43.81	17.52	4.20
3	15.85	110.07	16.56	4.89
4	15.68	31.15	15.68	4.97
5	19.79	64.19	20.50	4.20

Table 5: TWEAKER performance for optimally solved TWAVRP instances

TWEAKER finds solutions that on average are 1.86 per cent from the optimal solution in less than 5 seconds. This shows that TWEAKER is capable of finding near optimal solutions for TWAVRP instances.

9 Discussion and Conclusion

This thesis introduced an Ant Colony Optimization algorithm called TWEAKER to solve the Time Window Assignment Vehicle Routing Problem with uncertain demand. Demand variability was captured through the use of demand scenarios. The algorithm revolved around the definition of pheromone along the exogenous time windows of customers. Pheromone in this implementation can be viewed as a measure of how attractive it was for a vehicle to visit a customer at a certain moment of the day. The way in which pheromone is constructed allowed three ant colonies with three different objectives to work together. The results indicate that this set-up is effective in solving VRPTW problems. The results also indicate that TWEAKER may find a good time window assignment within a short amount of time but that it does not do so consistently. This may be due to the large number of model parameters that need to be set. It is more likely though that in such cases TWEAKER simply has not searched a sufficient area of the solution space. TWEAKER does find near optimal solutions for the TWAVRP instances of Spliet and Gabor (2012).
A venue for future research would be the investigation of time window assignments within clusters of customers. The previous section shows that TWEAKER performed less for the clustered problems than for the random and mixed problems. It would be interesting to find a way to improve how TWEAKER deals with clustered customers, which would make the algorithm much more effective.

References

[**1**] Angus, D. and Woodward, C. (2009). '*Multiple objective ant colony optimisation*,' Swarm Intelligence, 3, 69-85.

[**2**] Barán, B. and Schaerer, M. (2003). '*A Multiobjective Ant Colony System for Vehicle Routing Problem with Time Windows*,' in: Proceedings of the 21st IASTED International Conference on Applied Informatics, 97-102.

[**3**] Blum, C. (2005). '*Ant colony optimization: Introduction and recent trends*,' Physics of Life Reviews, 2, 353-373.

[**4**] Dengiz, B., Altiparmak, F. and Belgim, O. (2010). '*Design of reliable communication networks: A hybrid ant colony optimization algorithm*,' IIE Transactions, 42, 273-287.

[**5**] Dolnicar, S. and Leisch, F. (2010). '*Evaluation of structure and reproducibility of cluster solutions using the bootstrap*,' Marketing Letters, 21, 83-101.

[**6**] Dorigo, M. and Stützle, T. (2004). 'Ant Colony Optimization,' MIT Press: London.

[**7**] Ellabib, I., Calamai, P. and Basir, O. (2007). '*Exchange strategies for multiple ant Colony System*,' Information Sciences, 177, 1248-1264.

[**8**] Gambardella, L.M., Montemanni, R. and Weyland, D. (2012). '*Coupling ant colony systems with strong local searches*,' European Journal of Operational Research, 220, 831-843.

[**9**] Gambardella, L.M., Taillard, É., and Agazzi, G. (1999). '*MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows*'. In Corne, D., Dorigo, M. & Glover, F. (Eds.), New ideas in optimization (63-76). London: McGrawHill.

[**10**] Jia, S., Zhu, J., Yang, G., Yi., Jian, Du, B. (2012). '*A decomposition-based hierarchical optimization algorithm for hot rolling batch scheduling problem*,' International Journal of Advanced Manufacturing Technology, 61, 487-501.

[**11**] Potvin, J.-Y., Rousseau, J.-M. (1995). '*An Exchange Heuristic for Routeing Problems with Time Windows*,' Journal of the Operational Research Society, 46, 1433-1446.

[**12**] Pureza, V., Morabito, R. and Reimann, M. (2012). '*Vehicle routing with multiple deliverymen: Modeling and heuristic approaches for the VRPTW*,' European Journal of Operational Research, 218, 636-647.

[**13**] Reimann, M. and Ulrich, H. (2006). '*Comparing backhauling strategies in vehicle routing using Ant Colony Optimization*,' Central European Journal of Operations Research, 14, 105-123.

[**14**] Rizzoli, A.E., Montemanni, R., Lucibello, E., Gambardella, L.M. (2007). '*Ant colony optimization for real-world vehicle routing problems: From theory to applications*,' Swarm Intelligence, 1, 135-151.

[**15**] Socha, K. and Dorigo, M. (2008). '*Ant colony optimization for continuous domains*,' European Journal of Operational Research, 185, 115-1173.

[**16**] Solomon, M.M. (1987). '*Algorithms for the vehicle routing and scheduling problems with time window constraints*,' Operations Research, 35, 254-265.

[**17**] Spliet, R. and Gabor, A. (2012). '*The Time Window Assignment Vehicle Routing Problem*', working paper.

[**18**] Taillard, É, Gendreau, M., Guertin, F. and Potvin, J.-Y. (1997). '*A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows*,' Transportation Science, 31(2), 170-186.

[**19**] Yu, B., Yang, Z.Z. and Yao, B. Z. (2011). '*A hybrid algorithm for vehicle routing problem with time windows*,' Expert Systems with Applications, 38, 435-441.