Online Stochastic Vehicle Routing

by

Kasper Galschiøt Markus

Master's Thesis
Department of Mathematics and Computer Science (IMADA)
University of Southern Denmark
May 2009

Supervisor: Professor, Ph.D.,Dr. Scient., Jørgen Bang-Jensen

Abstract

This thesis deals with the Dynamic Vehicle Routing Problem (DVRP) for which stochastic information is available. DVRP is interesting in that it enables the modeling of many practical applications, that the offline VRP is not able to handle. Traditionally, the DVRP is solved in one of two ways: when customers become known throughout the run, oblivious online algorithms are used, in which the route is extended as more customers becomes available. Alternatively, when all customers are available but with uncertainties in their properties, stochastic optimization is used, which build the routing plan a priori, and then modifies it when changes in customer properties occur.

This thesis is based on Van Hentenryck and Bent [2006], which present a new approach to solving the DVRP by the use of stochastic information to guide the algorithm. The idea is to use this extra knowledge to make more enlightened decisions than oblivious online algorithms and furthermore handle more dynamic instances than stochastic optimization is able to solve efficiently.

The algorithms of Van Hentenryck and Bent [2006] are implemented and extended to efficiently handle continuous positionings of customers by means of discretization of the map. Furthermore, the relatively new offline algorithm ABHC is implemented as a sub-procedure for the algorithms. These algorithms are compared to three oblivious online algorithms as well as solutions found by an offline algorithm (ABHC). The results found are not entirely consistent with those of Van Hentenryck and Bent [2006].

In most cases, the use of stochastic knowledge seems to help the algorithms find better solutions than using oblivious online algorithms. Furthermore, the extension of the algorithms seems to improve the efficiency of the algorithms, but further testing is needed to confirm this.

Resumé

Dette speciale omhandler dynamisk ruteplanlægning, hvor stokastisk information om problemet er tilgængeligt. Dette er interessant, fordi det tillader modellering af mange praktiske applikationer som normal offline ruteplanlænging ikke er i stand til at håndtere.

Traditionelt løses dynamisk ruteplanlægning på én af to måder, alt efter karakteren af dynamik. Hvis det kun er en delmængde af kunderne, der kendes fra starten af dagen, bruges klassiske (uvidende) online algoritmer, der laver en ruteplan og så udvider den i takt med, at flere kunder bliver kendte. Hvis alle kunder derimod kendes, men med usikkerheder om deres egenskaber (såsom tidsvinduer), bruges stokastiske optimering, hvori en færdig ruteplan laves a priori, og så tilpasses de ændringerne når de forekommer.

Dette speciale er baseret på Van Hentenryck and Bent [2006], der præsenterer en ny tilgang til løse dynamiske ruteplanlægningsproblemer. Dette gøres ved at bruge den stokastiske viden, der er tilgængelig om problemet til at guide planlægningen af ruteplanen. Tanken er, at denne ekstra viden kan bruges til at tage mere oplyste, og dermed bedre, valg end de uvidende online algoritmer. Desuden er algoritmerne i stand til at håndtere mere dynamiske instanser end stokastisk optimering er i stand til effektivt at håndtere.

Algoritmerne fra Van Hentenryck and Bent [2006] er implementeret og udvidet til effektivt at kunne håndtere kontinuerte placeringer af kunder. Dette er gjort ved en diskretisering af kortet. Desuden er den relativt nye offline algoritme ABHC implementeret som en sub-procedure til algoritmerne. De implementerede algoritmer bliver sammenlignet med tre uvidende online algoritmer samt offline løsninger fundet af ABHC. Resultaterne fundet i dette speciale stemmer ikke fuldstændigt overens med resultaterne fra Van Hentenryck and Bent [2006].

I de fleste tilfælde virker det til, at brugen af stokastisk viden hjælper algoritmerne til at finde bedre løsninger end de uvidende online algoritmer. Desuden ser det ud til, at den implementerede udvidelse med diskretisering af kortet hjælper til at gøre algoritmerne mere effektive, men videre tests er nødvendige for at bekræfte det.

Acknowledgements:

Supervising:
Professor Jørgen Bang-Jensen

Random Good Advices:
M.Sc., Ph.D. Student, Steffen Elberg Godskesen

Proof Reading:
M.Sc. Anders Porsbo
M.Sc. Kristian Kirknæs Færk
B.Sc. Mette Wod

An Interesting Basis for a Master's Thesis:
Professor Pascal Van Hentenryck
Ph.D. Russell Bent

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Routing is of increasing importance in today's world. Whether it is browsing the Internet, sending mail, transporting goods or traveling, routing is involved. While it is possible to do this without efficient algorithms, significant economic, logistic and time wise gains can be achieved using the algorithms developed for this.

In Denmark, for example, transportation represents around 15% of national expenditures [Larsen, 1999]. Furthermore, it is estimated that distribution constitute almost half of the total logistics cost [De Backer et al., 1997]. Obviously, if the transportation expenditures of Denmark could be lowered by planning more efficient routes, yielding shorter traveling distance or less vehicles needed, significant economic gains could be made. Algorithms for the Vehicle Routing Problem (VRP) address the issue of finding an efficient routing plan, visiting customers, delivering goods, or something similar, while trying to minimize some objective (eg. travel distance). Informally, the VRP is defined as having a pool of customers that have to be visited once. A fleet of vehicles is available, and the problem is to visit the customers while minimizing some objective such as time, cost, etc.

Traditionally, a distinction between off- and online problems has been made. In offline problems all data is known, whereas online problems are oblivious to what data will appear during the execution. Naturally, there will be many problems that fall somewhere in between these two categories: where some or all customers are known, but with uncertainties that are not known a priori. In general these have been handled in two ways depending on the problem type. Either with dynamic optimization, where optimization typically is performed on the known customers and new are taken into account as they become available, or by stochastic optimization, in which some data is stochastic and an a priori solution is made, and then, if an unforeseen event occurs, a recourse function modifies the solution to handle this [Bent and Van Hentenryck, 2004a].

In terms of the VRP there are many applications in which the problem can be considered online. To name a few examples; the routing of police cars, taxi services or ambulances. Furthermore, there are even applications that would traditionally be considered offline but could be subject to sudden changes, like new customers appearing that urgently need service, traffic accidents making the planned routes impossible, changes in the demands of customers, changed traveling times due to heavy traffic, vehicles breaking down, etc.

During considerations on the subject of this thesis, Professor Jørgen Bang-Jensen recommended the book *"Online Stochastic Combinatorial Optimization"* by Van Hentenryck and Bent [2006], that presented a different approach to the problem of vehicle routing in a dynamic setting. Their approach is based on the fact that one, in general, has access to some stochastic knowledge of the problem properties through a probabilistic model of the problem or historical data. Van Hentenryck and Bent uses this knowledge to guide an online solver to good solutions by anticipating future events. The approach is very interesting and has yielded good results, compared to oblivious online algorithms.

To grab the details of this thesis, it is expected that the reader has a level of knowledge in the field of computer science at the level of masters degree or above.

1.1 Motivation

This thesis is based on, and motivated by the book of Van Hentenryck and Bent [2006], which amongst other is based on the work in [Bent and Van Hentenryck, 2003, 2004a,b,c,d,e; Bent et al., 2005; Bent and Van Hentenryck, 2005; Bent and Hentenryck, 2006]. Van Hentenryck and Bent describe a new way to handle the dynamic vehicle routing problem. Unlike previous stochastic algorithms, their methods are able to handle very dynamic instances. And unlike traditional dynamic algorithms, it bases its decision on other than current visible requests, by taking into account stochastic knowledge of the instance. While this means the algorithms need stochastic information, it is not unreasonable to assume available in many cases, since either stochastic models or, more commonly, historical knowledge of a problem instance often is available.

In the algorithms presented by Van Hentenryck and Bent [2006], they sample the expected amount of customers to appear during the run — that is, based on the stochastic knowledge of the instance, they make a qualified guess on which customers will appear and which properties they will have. Using these sampled customers along with the ones that are already visible at the current time constitutes a “sampled instance”. An offline heuristic can be used to solve this, yielding a routing plan. Repeating this multiple times, a pool of solved sampled instances becomes available. When a decision has to be made on which customer to serve next, this can be guided by the pool of plans, and in this way guiding the solution of the online problem by the stochastic knowledge available.

This is a very interesting way of handling the problem which, to a degree, has been explored previously by by Chang et al.. However, while Chang et al. achieved good results for online packet scheduling, their method is not well suited for the online stochastic VRP due to the limited time between decisions, and the computationally demanding optimization [Bent and Van Hentenryck, 2004c]. As documented in Van Hentenryck and Bent [2006], along with several of their articles, the methods they have developed achieves good results for the online stochastic VRP.

1.2 Aim of Thesis

The aim of this thesis is to implement the algorithms of Van Hentenryck and Bent for the VRP and to extend these. Furthermore, a comparison of the results with effective offline and online algorithms should be made, as to be able to evaluate the value of stochastic information when solving the dynamic vehicle routing problem.

When Van Hentenryck and Bent sample customers, this is done in a somewhat simplified way, in which customers can only appear in certain locations, with certain time windows, demand and service times. This will be explained further in Chapter 2, 3, and 4. In this thesis, this will be extended to allow customers to appear anywhere on the map, with arbitrary time windows, demand and service times according to some predefined distribution. To be able to use this efficiently with Van Hentenryck and Bents algorithms, a discretization of the map is used. The usefulness and effectiveness of such a discretization will be examined, and compared to the principles of Van Hentenryck and Bent.

While this will be the main goal of the thesis, the use of other offline algorithms, than the ones proposed by Van Hentenryck and Bent will be implemented, and their influence on the solution will be evaluated. Furthermore, different ways of sampling customers will be examined, like in Van Hentenryck and Bent [2006].

Finally, a summary of the implemented results is given along with ideas and discussions of further extensions and work.

2 Model

The Vehicle Routing Problem (VRP) is a well known and well studied problem. Due to the great amount of work that has been done on it, a great number of specializations have emerged, including capacity on vehicles, pickup and delivery of goods, time windows for customers, multiple depots, a multitude of objective functions, etc.. So before examining the algorithms developed for the Dynamic Vehicle Routing Problem, it is necessary to formulate the exact properties of the problem studied in this thesis as to avoid confusion on the problem being examined.

This chapter will start with an introduction to the most basic VRP. In section 2.2 and 2.3, these definitions will be extended to include capacity on vehicles and time windows. Finally in section 2.4, objectives relevant to this thesis will be discussed and defined, and in 2.5 a short note on the hardness of the problem is given.

Note that the VRP and the extensions defined here, are the ones needed to understand and define the offline algorithms dealt with in Chapter 6. The extensions and definitions related to the dynamic version of this problem will be examined in Chapter 3.

2.1 The Basic Vehicle Routing Problem (VRP)

One of the most classic, well known, and well studied problem in the field of optimization is the Traveling Salesman Problem (TSP). In this problem, a traveling salesman has to visit a set of cities while minimizing the overall traveled distance.

A generalization of this problem, is the Vehicle Routing Problem (VRP), in which we have several traveling vehicles (or in the terminology of TSP; several salesmen) who can visit the customers. The vehicles are identical and have to start and end their route in a depot. Each customer has a location and should be visited once by one vehicle. The problem is to serve all the customers while minimizing some objective (see section 2.4).

In the very basic version of VRP, we have a set $C = \bigcup_{i=1}^{n} c_i$ of customers that each has to be visited by one of m vehicles. Each customer must be visited exactly once by one vehicle. Therefore a customer is also called a request, and these two terms will be used interchangeably throughout the rest of this thesis. The vehicles depart from a depot o, and all routes must start and end here. Knowing that each vehicle has to start and end their route at the depot, serving customers in between, we can define a route as

$$\rho =< o, c_1, \ldots, c_n, o >, c_i \in C, c_i \neq c_j, \forall i, j \tag{2.1}$$

.

To denote the ordered set of customers in route ρ, $cust(\rho)$ is used. Each route is served by a vehicle, and in some contexts it is more natural to think of the route as a vehicle. For the remainder of this thesis, the words “route”, “tour” and “vehicle” will be used interchangeably depending on which word makes the understanding most clear.

We define R as the union of the customers and depot; $R = C \bigcup o$. Between each pair of neighbours in R a distance $d(c_i, c_j), \forall r_i, r_j \in R$ is defined. In the context of this thesis $d(c_i, c_j) = d(c_j, c_i)$. The total length of a route is the sum of the distance between the customers it contains; $d(\rho) = d(o, c_1) + d(c_1, c_2) + \ldots + d(c_n, o)$. It is possible to have a cost and/or travel time associated with each distance, but in the context of this thesis, these values are all considered the same.

A solution to the VRP is a set of routes serving the customers. This is called a routing plan (γ), and is defined

$$\gamma = (\rho_1, \ldots, \rho_m) \text{ where } \rho_a \neq \rho_b, \emptyset = \bigcap_{i=1}^{m} cust(p_i) \tag{2.2}$$

Together with (2.1), this definition ensures that no customer is served more than once. Like in the case of routes, $cust(\gamma)$ is used to denote the set of customers served by the routes of the routing plan;

$$cust(\gamma) = \bigcup_{i=1}^{m} cust(\rho_i)$$

Similarly, the length $d(\gamma)$ of a routing plan is the sum of the length of its routes:

$$d(\gamma) = \sum_{i=1}^{m} cust(\rho_i)$$

In a valid solution, the routing plan has to serve all the customers, which can be defined by the constraint:

$$R = cust(\gamma) \tag{2.3}$$

In summary the Vehicle Routing Problem is, given a set $R = C \bigcup o$ of vertices (customers and depot), and a set of edges between these:

$$d(r_i, r_j) \ \forall r_i, r_j \in R$$

find γ a routing plan minimizing the objective w:

$$\min w(\gamma) \tag{2.4}$$

subject to the constraint of equations (2.1), (2.2) and (2.3).

There are different options for the objective w. These will be explained in section 2.4 on page 8.

Defined above is the most basic VRP, but as mentioned, several variations exist. In the following sections, the ones relevant to this paper will be described.

2.2 The Capacitated VRP

In many cases of vehicle routing delivery of goods is involved, and to model this, the capacitated vehicle routing problem (CVRP) is used. As an example, a postal office needs to deliver mail to a set of customers, which could be done by having a fleet of vehicles visiting these customers and delivering the mail. Obviously the vehicles can only carry a certain amount of mail, thus the number of customers on a route could potentially be limited by the car not being able to carry any more mail — that is, the demand of the customers on a route must not surpass the capacity of the vehicle.

Formally, in CVRP, each customer c_i has a certain demand $q(c_i)$, and each vehicle is identical with a capacity of Q. We define the demand of a route to be:

$$q(\rho) = \sum_{c_i \in cust(\rho)} q(c_i)$$

that is, to be the accumulated demand of customers in route ρ. To model the constraint that capacity of the vehicle imposes on a route, an extra constraint is added to the problem:

$$q(\rho) \leq Q, \forall \rho \in \gamma \tag{2.5}$$

It is possible generalize the CVRP to the regular VRP. If we have $q(c_i) = 0, \forall c_i \in C$, the routes are no longer restricted by the demand, and we have the VRP problem. For more detailed information on the CVRP, see Toth and Vigo [2002].

2.3 VRP with Time Windows

An important and highly usable extension to the CVRP is the use of time windows (VRPTW) [Toth and Vigo, 2002]. These are used when one wants to model that a customer needs to be visited within a certain time span, or one wishes to set a deadline for when the vehicles has to return to the depot.

In VRPTW, each customer c_i has a service time $p(c_i)$, describing how long it takes to service him. The service time of the depot is $p(o) = 0$. To model the actual window of time in which a customer requires service, two variables $e(c_i)$ and $l(c_i)$ are defined $\forall c_i \in R$ modeling earliest and latest start of service, respectively. A car is allowed to arrive to a customer before $e(c_i)$, but servicing the customer cannot start until $e(c_i)$. The latest time that a vehicle is allowed to arrive at a customer is $l(c_i)$. Like customers, the depot also has a time window $e(o)$ and $l(o)$ assigned to it, but these have a somewhat different meaning. The vehicles may not depart from the depot before $e(o)$, and have to be back at the depot no later than $l(o)$. The time window for the depot allows to limit the total allowed time of a tour, and hereby for example modeling a working day from 8–16. The total time horizon for an instance is defined as $h = l(o) - e(o)$.

The time windows can be handled as soft or hard constraints; called soft and strict time windows, respectively. In the soft version, it is allowed to violate the time windows, but at a cost in the evaluation function. When using strict time windows, it is forbidden to break them. In this thesis time windows will be handled as hard constraints.

Like in the case of capacity, VRPTW can be generalized to the basic VRP. This is done by setting $e(c_i) = 0$ and $l(c_i) = \infty, \forall c_i \in R$. As specified in section 2.1, R is the union of all customers and the depot. Letting all customers have infinitely long time windows ensures that they will have no effect on the routing plan.

The VRP examined in this paper includes both of the extensions described above. That is, each customer has a certain demand, along with a time window in which service is allowed.

Beside extending the problem with different types of constraints, VRP also varies in what the objective of the problem is. The objectives relevant to this thesis are examined in the next section.

2.4 Objectives

Since vehicle routing is applicable in many different scenarios, the focus of the optimization varies. To model the goal of the routing plan, besides fulfilling the problem constraints, an objective function is used. The objective is, as mentioned above, to minimize the objective funtion:

$$\min w(\gamma)$$

Minimizing Route Length: A very common objective function is the minimization of the total length of the routing plan. This is relevant when one only wants to lower the travel cost or travel time and no consideration for vehicle employment is done Remembering the definition above, the length of a routing plan is $d(\gamma)$, so the objective function is defined as:

$$w_0(\gamma) = d(\gamma) \tag{2.6}$$

Minimizing Employed Vehicles then Length: A significant cost for companies for whom vehicle routing is relevant, is the salary to drivers, purchasing and maintenance of cars, and similar expenses. This makes limiting the number of employed cars relevant, and introduces another widely used objective function. Here, the primary objective is to minimize the number of vehicles and minimizing the length of the routing plan is the secondary objective. Letting $|\gamma|$ define the number of routes in the routing plan, the objective function can be modeled as follows:

$$w_1(\gamma) = a \cdot |\gamma| + d(\gamma) \tag{2.7}$$

where the constant a is the cost for employing a vehicle. Sometimes, it might be desirable to have priority on minimizing the number of vehicles, but only to a certain point, at which the decrease in route length makes up for the cost of adding another vehicle. This can be modeled by setting a to a cost, allowing the desired relationship between number of vehicles and length of routing plan. If the desired objective is to minimize $|\gamma|$ disregarding length of the routing plan, a should be set sufficiently large to render the length irrelevant in all other cases than the comparison of two routing plans with an equal number of employed vehicles used. More precisely, this should be modeled as a lexicographic function which has to be minimized:

$$w_{1,1}(\gamma) = \big(|\gamma|, d(\gamma)\big) \tag{2.8}$$

Minimizing Unserved Customers then Length: A less commonly used, but highly realistic objective, is one in which we have a limited number of vehicles, and the focus is on maximizing the number of customers served. This is used, amongst other, by Van Hentenryck and Bent [2006]. In this case, it might not be possible to serve all the customers. As mentioned the objective here is to serve as many customers as possible, and secondly minimize the route length. An example of a service in which this could be relevant is taxi-companies. These have a fixed number of vehicles available, and on some occasions it is not possible to serve all customers due to the limited amount of vehicles (eg. new years eve). Note that when using this objective, the constraint (2.3) is not used, since we want it to be legal (although undesirable) to not serve some customers. Modeling this objective is very similar to objective (2.7):

$$w_2(\gamma) = a \cdot (-|cust(\gamma)|) + d(\gamma) \tag{2.9}$$

Note that we minimize the negative cardinality of the customer set $cust(\gamma)$, which equals maximizing the number of customers in that set. Again, a constant a is used to specify the priority of the served customers over route length. And, as with $w_1(\gamma)$ from (2.7), making a sufficiently large, renders the distance irrelevant unless the number of customers are the same in the two routing plans. Again, a more correct way of modeling this would be with the lexicographic function:

$$w_{2,1}(\gamma) = \big(-|cust(\gamma)|, d(\gamma)\big) \tag{2.10}$$

2.5 Hardness

The VRPTW treated in this thesis can, as described above, be generalized to CVRP by setting $e(c) = 0$ and $l(c) = \infty, \forall c \in R$. The CVRP can be generalized to VRP if $q(c) = 0, \forall c \in R$, which in turn can be generalized to the Traveling Salesman Problem when the number of vehicles available is only 1. The TSP has been proven NP-complete [Cormen and Stein, 2001] which implies by restriction that VRP, CVRP and VRPTW are NP-complete. Under the assumption that $NP \neq P$, this means that the problem is not solvable in polynomial time. While it has neither been proven nor disproven that $NP \neq P$, it is a common belief that this is the case.

It should also be emphasized that the VRPTW is a combinatorial optimization problem which is extremely hard to solve [Van Hentenryck and Bent, 2006]. As an example, the Solomon benchmarks [Solomon, 1987] are very well known and contains a maximum of only 100 customers. Although considerable research in VRPTW has been made and most algorithms, some very elaborate, are tested against these, optimal solutions still have not been found for several of them.

3 The Dynamic Vehicle Routing Problem

In Chapter 2, the standard offline problem of Vehicle Routing was described. Being offline, it is assumed that all information about the problem is known beforehand, but this is not always possible. Taxi services, package pickup and deliveries, police vehicles, ambulances and many more, do not know all the destinations they have to visit, let alone travel times, time windows or demands. If not all the information is available from the start, the problem is defined as dynamic. In a paper of Psaraftis [1988], the following definition is given: “A vehicle routing problem is static if the inputs to the problem do not change, neither during execution of the algorithm that solves it nor during the execution of the solution; a problem is considered dynamic when inputs to the problem become known to the decision maker or are updated concurrently with determination of the solution.”

This simple definition simply states that some input, initially unknown, has to become known or updated during the run, but allows a great deal of freedom in the amount of input that has to be dynamic. To be dynamic, it is not necessary that all the properties of VRP are dynamic. Indeed, most studies carried out on the Dynamic Vehicle Routing Problem (DVRP) researches versions in which only a specific part of the input data is dynamic, this could be travel times, service time, demand, etc..

It is relevant to examine the DVRP, because in real life, it is common that parts of the problem are unknown until execution of the solution. Travel times between two points for example, will almost always be dynamic, due to traffic, weather conditions or similar, although these variations in time might be insignificant enough, to simply ignore them and consider the problem static (offline). Accepting the fact that part of the problem is dynamic, allows us to create algorithms that are more suitable for solving real life instances. Lots of research has been done in the DVRP, as far back as 1976, when W.R. Stewart presented the Delivery Truck Routing Problem with Stochastic Demands [Stewart, 1976], and next in Cook and Russell [1978], where the authors examined a stochastic VRPTW.

Realistically, if the exact inputs of the VRP are unknown, it is common to have some idea of their distribution. Taking the example of travel times, it might not be possible to know the exact times it takes to travel between two points, but it would be often be the case that one knows what times a day the traffic is heavy or scarce, yielding longer or shorter travel times, respectively. Having this knowledge of future events, by their distribution or approximations hereof, makes the problem stochastic [Hvattum et al., 2007]. Obviously the stochastic VRP is a subtype of DVRP, since not all inputs are known (precisely) until during the execution. Using this stochastic knowledge has proven useful in guiding the algorithm towards good solutions.

Before looking at the advantages, and general handling, of stochastic information as a guide for the algorithms, there are some important properties that need to be taken into account when switching to a dynamic setting. These properties will be examined in the remaining part of this chapter. In Chapter 4, the use of stochastic information to solve the dynamic VRP will be examined. In chapter 7, online algorithms that use no stochastic knowledge will be discussed.

3.1 Switching to a Dynamic Setting

Since some variables will be unknown in a part of the execution, it is useful to keep track on some additional information such as the slack between time windows and which customers have already been visited.

Keeping track of the slack in time windows, showing how much it is possible to shift a visit to a customer without violating its time windows, can be helpful in quickly determining if the prolonging of the current visit, sudden insertion of a new customer, or the like, is feasible. Keeping track of slack will be examined in section 3.2.

Due to the fact that the travel time, service time, demand, or whichever variables are dynamic, may become known or change over the course of execution, the part of the routing plan that comes after the current time t is uncertain. On the other hand, the part of the routes visited before t has become static, since they are in the past, and cannot be changed. The handling of fixating part of the routing plan is examined in section 3.3. Finally, some useful functions, in context of the DVRP will be defined in 3.4.

3.2 Defining Slack: Earliest Departure (δ) and Latest Arrival (z)

When using time windows in a dynamic setting, it might be a good idea to keep two more variables associated with each customer; earliest departure δ and latest arrival z. During the run of an algorithm, it is nice to know how much slack there is for each customer, ie. what is the latest and earliest time the customer can be served without resulting in any violation of the time window of the other customers of the route.

The earliest departure δ for a customer c, as the name suggests, is the earliest time the schedule allows for the vehicle to finish servicing that customer. Obviously this variable is based on the

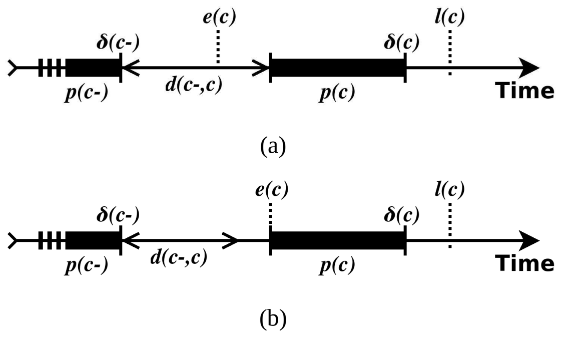


Figure 3.1: Illustrations of Earliest Departure $\delta(c)$: Two examples of earliest departures for a customer c.

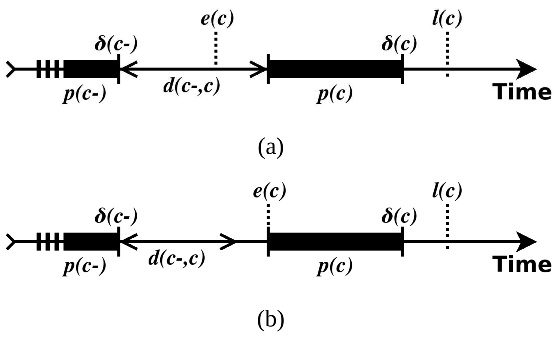


Figure 3.2: Illustrations of Latest Arrival $z(c)$: Two examples of latest arrivals for a customer c.

route prior to customer c and $e(c)$. Letting c^{-} denote the customer prior to c, the recursive formula for earliest departure is given by:

$$\begin{aligned}&\delta(o) = e(o)\\&\delta(c) = \max\big(\delta(c^{-}) + d(c^{-}, c), e(c)\big) + p(c),\ c \in cust(\gamma)\end{aligned} \tag{3.1}$$

For the depot, which has a service time of 0, the earliest possible time to depart is the same as the start of service $e(o)$. For customers the earliest start of a service for c depends on two things. Either the previous customer was done early enough for the vehicle to reach customer c before or at $e(c)$ in which case service can start at $e(c)$. If not the service can start as soon as the vehicle has left the previous customer (c^{-}) and reached customer c. It takes $p(c)$ to serve customer c, yielding $\delta(c)$. Figure 3.1 shows the two possible cases for $\delta(c)$.

Similarly, the latest arrival z takes into account all the customers on the route following customer c, to know what time the vehicle can arrive at the latest, and still be able to serve customer c without causing violations later in the route. Since we know the time that the vehicles have to return to the depot ($l(o)$), the latest arrival time can be recursively defined as:

$$\begin{aligned}&z(o) = l(o)\\&z(c) = \min(z(c^{+}) - d(c, c^{+}) - p(c), l(c)), c \in cust(\gamma)\end{aligned} \tag{3.2}$$

where the first part of the minimum function makes sure the customer c^{+} following c will be reached before its latest arrival time, and the second part ensures that the time window for customer c is not violated. Figure 3.2 shows two scenarios for calculation of $z(c)$.

3.3 Fixating Served Customers

When dealing with a DVRP, it is desirable to change the routing plan during execution of the algorithm, to take the new/updated variables into account. To do this, it is necessary to keep

Algorithm 1: insertable

```
Data: Customer to be inserted ($c_{ins}$), customer immediately preceding insertion point
      ($c_{pos}$), route to insert into ($\rho$)
Result: True if insertable, False otherwise
if $q(c_{ins})+q(\rho) > Q$ then
    return false
else if fixated($c^+_{pos}$) then
    return false
else
    $\delta(c_{ins}) \leftarrow$ calc $\delta$ as if $c_{ins}$ was placed after $c_{pos}$
    $z(c_{ins}) \leftarrow$ calc $z$ as if $c_{ins}$ was placed after $c_{pos}$
    slackNext $\leftarrow z(c^+_{pos})$-($\delta(c_{ins})+d(c_{ins}, c^+_{pos})$)
    slackPrev $\leftarrow z(c_{ins})$-($\delta(c_{pos})+d(c_{pos}, c_{ins})$)
    return slackPrev $> 0 \bigwedge$ slackNext $> 0$
end
```

track of which customers have already been served, since only the part of the routes beyond this point is changeable.

This can be done by associating a flag with each customer, defining whether it is fixated (ie. immutable) in which case, it is not possible to change the time a vehicle visits it. We use $fixated(c) = true$ to define a customer as fixated. At any given time t, the customers for which service has started will be fixated, and it is therefore not possible to move these to a different place in route, or inserting new customers before them. If a customer c is fixated, all customers c_x preceding this will have $fixated(c_x) = true$.

If a customer has already been served, we know its exact time of visit, and therefore $z(c)$ and $\delta(c)$ become static, reflecting the visiting time of the customer.

3.4 Other Useful Definitions and Functions

Having defined the variables $z(c)$, $\delta(c)$ and $fixated(c)$, we can now define functions for checking whether an insertion of a customer is possible, calculating the actual insertion, and calculating the effect of removing a customer.

Checking for Insertion: One common functionality needed is to check whether it is possible to insert a customer at a certain position in the route. The algorithm for this is shown in Algorithm 1. As input, the customer to insert (c_{ins}) is given, along with the customer immediately preceding the insertion point (c_{pos}) and the route ρ. It is assumed that c_{ins} is an unrouted (and not fixated) customer. The first two lines simply tests that the capacity of the vehicle is not surpassed when inserting the new customer. c^+_{pos} is the customer following the insertion point. As described above, if c^+_{pos} is fixated, this means that we have already visited it, and hence cannot change the route before this point, yielding an infeasible insert (line 3 and 4). If this is not

Algorithm 2: updateEarliestDeparture

Data: Customer c

Result: Updated values of earliest departures (δ) from customer c and onwards until not necessary, or depot is reached

```
1 if $c \neq o$ then
2     oldED ← $\delta(c)$
3     $\delta(c)$ ← p($c$)+(max($\delta(c^-)$+d($c^-, c$),e($c$))
4     if oldED ≠ $\delta(c)$ then
5         updateEarliestDeparture($c^+$)
6     end
7 end
```

the case, we examine whether the time windows allows for the insert. In line 5 and 6 $\delta(c_{ins})$ and $z(c_{ins})$ are calculated, as if the customer was inserted. Recall that $z(c)$ defines the latest arrival allowed at customer c, while keeping the following part of the route feasible. $\delta(c)$ is the earliest departure possible for customer c based on the previous part of the route. Line 6 along with line 8 and the second part of the if-statement of line 10, checks that the earliest possible arrival at c^+_{pos} is no later than the latest allowed arrival. In the same way, line 7 and 9, along with the first part of the if-statement checks that the earliest possible arrival at the new customer, is no later than the latest arrival allowed. If these requirements are fulfilled, the insertion is legal.

Updating δ and z: Before examining the algorithms for removing and inserting customers, we need to look at two important sub procedures, namely the updating of δ and z. As can be seen from their definitions in equations (3.1) and (3.2), changes in the route is likely to affect them.

In Algorithm 2, the algorithm for updating the earliest departure (δ) is given. Given equation (3.1), the algorithm is almost self explanatory, except for a few points. In line 2, we save the old earliest departure of the customer. Line 3 calculates the new δ, and the two values are compared in line 4. If they differ, the following customers δ need to be updated by a recursive call to `updateEarliestDeparture`. The reason for this comparison, is that if the δ-value of customer c is not changed, neither will the δ value of any of the following customers, since they depend on $\delta(c)$, and are assumed valid before this algorithm. This might save some computation time.

Algorithm 3 shows the algorithm for updating the latest arrival (z). Besides calculating a different value, this algorithm is very similar to 2, and straightforward. Again, line 5 checks whether the value of $z(c)$ has changed. In case it has not, it is not necessary to update the z-value of any of the previous customers.

Insertion and Removal: Having defined the algorithms for updating δ and z, we can now look at the insertion and removal of a customer given in Algorithm 4 and 5, respectively.

For insertion, we simply place the customer on the desired position, and update its δ and z values. Since the functions for updating these values were recursively defined, the update will

Algorithm 3: updateLatestArrival

```
Data: Customer $c$
Result: Updated values of latest arrival ($z$) from customer $c$ and onwards until not
        necessary, or depot is reached
1 if fixated($c$) $=$ False then
2     if $c \neq o$ then
3         oldLA $\leftarrow z(c)$
4         $z(c) \leftarrow \min(z(c^+)$-$d(c, c^+)$-$p(c), l(c))$
5         if oldLA $\neq z(c)$ then
6             updateLatestArrival($c^-$)
7         end
8     end
9 end
```

Algorithm 4: insert

```
Data: Customer to be inserted ($c_{ins}$), customer before insertion point ($c_{pos}$)
1 place $c_{ins}$ between $c_{pos}$ and $c^+_{pos}$
2 updateEarliestDeparture($c_{ins}$)
3 updateLatestArrival($c_{ins}$)
```

Algorithm 5: remove

```
Data: Customer to be removed ($c$)
1 remove $c$ from between $c^-$ and $c^+$
2 updateEarliestDeparture($c^-$)
3 updateLatestArrival($c^+$)
```

propagate to the relevant customers. It is assumed that the insertion is checked for validity (by Algorithm 1) before a call to insertion is made.

The removal algorithm is just as simple. Here, the customer is removed from between c^- and c^+, and their δ and z values are updated. The reason the call to `updateEarliestDeparture` is only done on c^-, is that the update will propagate to c^+ via the functions recursive nature. The same holds for `updateLatestArrival` and c^+.

The functions described in this section is used as sub-procedures for many of the algorithms implemented in this thesis. The actual selection of, for example, a customer to insert (c_{ins}) as well as a suitable insertion point (c_{pos}) is handled by the algorithm using these sub-procedures.

4 Online Stochastic Algorithms

As described above, one way to handle the DVRP is by online algorithms. This is indeed a realistic approach and clever algorithms are developed that allow for finding good solutions. But often one is not completely oblivious of the future events which the oblivious online algorithms fail to take advantage of. Stochastic knowledge can often be made available, which is what Van Hentenryck and Bent use to guide in their algorithms presented in this chapter. When future events are known to some extent by their probability distributions, the dynamic problem is defined to be stochastic [Hvattum et al., 2007].

While one might not have an exact stochastic model, some stochastic knowledge can be derived from historical data, or through the use of a Hidden Markov Model. Whatever method one uses, having these probabilities available can be of great help when creating the routing plan. The weakness of the oblivious online algorithm, is that it does not have anything to base its decisions on, except for the requests that have been made at a given point in time, and the routing plan generated thus far. When having a stochastic problem, one can use the knowledge of the variables to attempt to predict future requests, and use this to guide the algorithm to better solutions, which is exactly what Van Hentenryck et al. does.

It is easy to think up cases in which some sort of knowledge of future events are probable. When routing police vehicles, often some neighborhoods are more dense in terms of crime, more crime is committed during the night, and wealthy neighborhoods are subject increased amounts of break-ins, and even more so during holidays. For a package delivery company, industrial neighborhoods will have many requests during the working hours, and certain big companies will be more likely to request service than smaller companies.

A problem can be stochastic in several ways, depending on which variables are dynamic (and known by their probability distributions). For the problem treated in this paper, customers arrive dynamically as the algorithm proceeds. When a customer appears, all its variables, ie. time windows, location, demand and service time, are known. For this thesis, stochastic knowledge of all variables of requests is assumed available.

4.1 Algorithm Overview

In the Online Stochastic Algorithms presented by Van Hentenryck and Bent [2006] and the articles referred to in section 1.1, the stochastic knowledge mentioned above is utilized in different ways to achieve good online solutions to the stochastic VRPTW. In this section an introductory overview of the main algorithm will be given, along with a short description of the extensions to this and a reference to the sections where these are described.

In the main algorithm, a partial “master” plan is kept which is fixated up to the current time t. At all times, the algorithm also has a set of visible but unrouted customers. Based on some stochastic knowledge, the customers of the remaining time horizon $]t; h]$ can be sampled. This is done utilizing the stochastic knowledge, an estimate of the total number of expected customers, and the customers currently visible. These sampled customers, visible unserved customers, and partial plan make up a “sampled instance”. This sampled instance can be solved by a regular offline algorithm, which of course should be modified to only change the parts of the plan which is in the future, ie. the customers not in the partial plan. By generating, and solving several of these sampled instances whenever time is available, the algorithm continuously has a pool of sampled instances available. When a vehicle is idle, due to having finished service of a customer, the decision of which customer to add to the partial plan for the idle vehicle is based on the pool of sampled routing plans. When the next customer for the vehicle is chosen, the pool is pruned for plans not compatible with the partial plan. In this way the construction of the route is guided by the sampled plans.

The generic algorithm for this is described in section 4.2, and variations of sub-procedures are described in the following sections. Section 4.3 and 4.4 describe different ways of choosing the next customer to serve; the Consensus [Bent and Van Hentenryck, 2004b] algorithm in section 4.3 and the Regret [Bent and Van Hentenryck, 2004c] algorithms in 4.4.

For the basic Regret and Consensus algorithms, only real customers can be selected to be served next. Van Hentenryck and Bent presents two different strategies for allowing the selection of sampled customers as next, called waiting and relocation. These strategies are compatible with both Consensus and Regret, and are described in section 4.5.

As described in Chapter 1, one of the aims of this thesis is the extension of Van Hentenryck and Bents model to allow arbitrary positions and properties of the sampled customers. This is described in section 4.6, and the method for pruning of customers is described in section 4.7. The idea of sampled customers, is that they represent potential real customers. Therefore, if a materialization of a sampled customer becomes available, this should of course be serviced instead of the sampled customer. This is described in section 4.8.

4.2 Generic Online Stochastic Routing Algorithm

In this section the general stochastic online algorithm is outlined and the essential parts of it are explained. Some of the sub-procedures will require more detailed explanations, while others have several options for implementations. These will be discussed in the following sections, rather than here.

The outline of the General Stochastic Online algorithm can be found in Algorithm 6. The algorithm starts by initializing the partial plan to empty (line 1), and retrieving the requests available from the start (line 2). Based on these, along with stochastic knowledge of the customers, a pool of plans is generated containing a mixture of sampled and real customers. This is done in the `generateSolutions` function, detailed in Algorithm 7.

In line 4, the main loop is entered, stepping one unit in time for each iteration. First, the partial plan is fixated up until the current time t. Following this, the newly visible customers are retrieved (line 7), and if one or more are materializations of sampled customers in the partial plan, this materialization is done. This is followed by a pruning of the plans that are not compatible with this materialization (line 8-12). A discussion of the materialization of customers is done in

Algorithm 6: Generic Online Stochastic Algorithm

```
   Result: Full Plan $\gamma_h$
1  $\gamma_0 \leftarrow$ empty plan
2  $R_0 \leftarrow$ customers available at start
3  $\Gamma \leftarrow$ generateSolutions($\gamma_o$, $R_0$, $t$, $h$) //See Algorithm 7
4  for $t \leftarrow 1$ to $h$ do
5      $\gamma_t \leftarrow \gamma_{t-1}$
6      $\gamma_t \leftarrow$ fixUntill($t$)
7      $R_t \leftarrow$ customers becoming available at time $t$
8      if materializable($R_t$) then
9          $\gamma_t \leftarrow$ materialize($R_t$)
10         $R_t \leftarrow R_t \setminus$ materialized($R_t$)
11         $\Gamma \leftarrow$ prune($\Gamma$,$\gamma_t$)
12     end
13     $P_{id} \leftarrow$ getIdles($\gamma_t$)
14     if $P_{id} \notin \emptyset$ then
15         $ts \leftarrow$chooseRequest($\gamma_t$, $\Gamma$)
16         for $i \leftarrow 1$ to number of vehicles $m$ do
17             if $\rho_i \in P_{id}$ then
18                 $\rho_i \leftarrow \rho_i : ts_i$
19             end
20         end
21         $\Gamma \leftarrow$ prune($\Gamma$, $\gamma_t$)
22     end
23     $\Gamma \leftarrow$ generateSolutions($\gamma_t$, $R_t$, $t$, $h$)
24 end
```

section 4.8.

If a vehicle has no (more) customers assigned or when it has finished service at a customer, it is considered idle. When a vehicle is idle, a decision will have to be made on which customer to serve next. Line 13 and 14 checks whether any vehicles are idle. If this is the case, the decision of which customers should be assigned to the vehicles are made by a call to `chooseRequest` (line 15). This is based on the current plan (γ_t) and the pool of sampled plans Γ. Van Hentenryck and Bent present multiple ways of deciding which customers to serve next, and this is described in detail in the following sections (section 4.3-4.5).

Line 16-20 extends the routes of idle vehicles with the customers selected in `chooseRequest`. The pool of sampled plans is updated in line 21, by pruning plans that are not compatible with the new requests. The mechanics of the pruning is described in more detail in section 4.7.

Finally, in line 23, more sampled plans are added to the pool of sampled solutions for the remaining time by a call to `generateSolutions` (Algorithm 7).

In Algorithm 7, the remaining customers are sampled (line 3), and the resulting instance is solved using some offline algorithm $\mathcal{O}$, and the result is added to the pool of solved sampled

Algorithm 7: GenerateSolutions

```
Data: Partial plan $\gamma_t$, visible unserved customers $R_t$, current time $t$, time horizon $h$
Result: Pool of solved sampled instances $\Gamma$
1 $\Gamma \leftarrow \emptyset$
2 repeat
3     $A \leftarrow R_t$ :sample($t, h$)
4     $\Gamma \leftarrow \Gamma \bigcup \mathcal{O}(\gamma_t, A)$
5 until time t+1
```

instances (line 4). This is repeated as long as time is available.

4.3 Choosing Requests by Consensus, chooseRequest-$\mathcal{C}$

We now consider one of the algorithms for choosing which customers to serve next, when a vehicle is idle. This algorithm is called Consensus ($\mathcal{C}$), and was presented in the article Bent and Van Hentenryck [2004b]. Consensus functionality is achieved through implementation of the `chooseRequest` function in the general stochastic online algorithm given in Algorithm 6.

The main idea is to choose the customer which most of the sampled routing plans agree on — or rather, choose a customer based on consensus of the sampled plans. The basic idea is; for each of the routing plans, find the customers it serves next on each route, and increment a counter for each of these customers by one. After having looked at all the sampled routes, the higher the counter of a customer, the more consensus there is of placing it next on a route. In other words, the mode of the customers to serve next on each route is found. While the principle is very simple, there are some things to take into consideration, such as ensuring the resulting customer selections are feasible, how to handle sampled customers and how to break ties.

A more detailed explanation of the algorithm along with at discussion of the considerations mentioned above is given below, in section 4.3.1.

4.3.1 Algorithm Outline and Explanation

An outline of the algorithm is given in Algorithm 8. In the first line, a set F of all visible requests at the current time t is created. Line 2-4 initializes a counter f for each of these request to 0. The `firstNonFixatedReal` function, given a routing plan, returns the first non-fixated *real* (ie. not sampled) customer for each route. These customers are saved in the set tr. For each of these customers, the score is increased by 1 (line 7-9). This is done for all routing plans (line 5-10). In summary each decision of customer is evaluated individually and independent of the vehicles they are assigned to. Finally, in line 11, the plan which has the set of `firstNonFixatedReal` that sums up to the highest score is found, and its `firstNonFixatedReal` customers are returned (line 12).

A more extensive explanation of line 11 might be suitable. Each plan γ is evaluated by summing the evaluation of its next first non-fixated real customers $(r_1, ...r_m)$:

Algorithm 8: chooseRequest-$\mathcal{C}$

```
Data: Sampled routing plans (Γ)
Result: List of consensus customers
1  F ← $\bigcup_{i=0}^{t} R_i$
2  foreach $r \in F$ do
3      $f(r) \leftarrow 0$
4  end
5  foreach $\gamma \in \Gamma$ do
6      $tr \leftarrow$firstNonFixatedReal($\gamma$)
7      foreach $tr_i \in tr$ do
8          $f(tr_i) \leftarrow f(tr_i) + 1$
9      end
10 end
11 $\gamma^c \leftarrow \arg\max(\gamma \in \Gamma) \sum_{i=1}^{m} f($firstNonFixatedReal$(\gamma)_i)$
12 return firstNonFixatedReal($\gamma^c$)
```

$$f(\gamma) = \sum_{i=1}^{m} f(r_i)$$

The plan γ that has the highest value of $f(\gamma)$ is selected as being the best.

There are several important points to be made in connection to the Consensus algorithm. First of all it is noteworthy how the sampled customers of the plans are handled. When looking at which customers each plan has set to be served next, only the real customers are considered. This is done via the `firstNonFixatedReal` function, and ensures that consensus can only be on real customers. There are also alternative strategies, where sampled customers are not being ignored. These will be described in section 4.5. It might seem illogical to evaluate customers independently of routes, and then selecting the plan with the highest score. Instead, one could evaluate customers vehicle wise. Then return the consensus for each vehicle as being the customer with the highest score that appears as first non-fixated real customer in that route. The reason for not doing so is to avoid infeasibility. For example, the same customer could be chosen as consensus for two different routes, which would be both undesirable and infeasible. By evaluating entire plans, it is ensured that a overall good, and feasible, selection of consensus customers is made. A final thing to point out, is how ties in line 11 is handled if two or more plans have the same total score. This is handled by selecting one randomly.

There are some problems with the Consensus algorithm. It is qualitative, in the sense that it increases the evaluation f of a customer by one, and ignores the actual score of the solution the customer is part of. It is also elitist, in that it only credits the best requests, while all others are ignored. The qualitativeness can be remedied by increasing the evaluation of a customer by the objective function it is part of. While this makes it quantitative, according to Bent and Van Hentenryck [2004c], this results in favoring the best requests even more, contributing to the algorithms elitism which is undesirable. The problem with elitism, is that several requests might be almost equal in quality, but only the very best receives credit. Furthermore, and more importantly, a request may not be the best for any sample, but might be very robust overall. Attempts to address these issues were made with the Regret algorithm, described in the following section.

4.4 Choosing Requests by Regret, chooseRequest-$\mathcal{R}$

The main motivation for the Regret algorithm [Bent and Van Hentenryck, 2004c], was to address the issues the Consensus algorithm had with elitism. Only increasing the evaluation of the best requests is problematic, in that it does not consider almost equally good requests, or the overall robustness of a request. This is attempted remedied by introducing the regret of a request r. That is, an aprroximation of the difference in quality when choosing r rather than the best customer of the vehicle i, ie. `firstNonFixatedReal`$_i$. In many applications, this can be done relatively fast, and the possible gain in selecting a suitable consensus might be worth the extra computations. A more detailed description and algorithm outline will be given in the following section.

4.4.1 Algorithm Outline and Explanation

An outline of the Regret algorithm is given in Algorithm 9. The algorithm is exactly the same as the Consensus algorithm given above, except for lines 9-11, in which the evaluation of the regret customers is made. Here, the set of customers for which to calculate regret is considered. For each customer r in this set, if it is possible to interchange r and the `firstNonFixatedReal` of the current route (tr_i), the evaluation of r is incremented (line 11). As suggested in by Bent and Van Hentenryck [2004c], the increase in evaluation for the regret customer r is 1 if a feasible swap can be found, and 0 if not. In this way, it is recognized that some choices of customers are equivalent, and furthermore, flexibility is awarded, by awarding customers that can feasibly be swapped, thereby eliminating some of the elitism from the Consensus algorithm.

It still remains to specify for which customers to calculate regret. Although the Regret algorithm has been detailed in several articles, it is unclear exactly which customers are subject to regret calculation. There are several options, and these will be described in the following.

In Bent and Van Hentenryck [2004c], the following explanation on regret is given;

> Consider the decision of choosing which customer to serve next on vehicle v and let s be the first customer on the route ρ of vehicle v. To evaluate the regret of another customer r on a vehicle v, the key idea is to determine if there is a feasible swap of r and s on v, in which case the regret is zero. Otherwise, if such a swap violates the time window constraints, the regret is 1.

This could indicate that only customers on the same route are considered. Defining ρ_u to be the part of the current sampled route that is not fixated, the set Re of regret customers would be:

$$Re = \rho_u \bigcap F \tag{4.1}$$

Algorithm 9: chooseRequest-$\mathcal{R}$

```
Data: Sampled routing plans (Γ)
Result: List of consensus customers
1  F ← $\bigcup_{i=0}^{t} R_i$
2  foreach $r \in F$ do
3      $f(r) \leftarrow 0$
4  end
5  foreach $\gamma \in \Gamma$ do
6      $tr \leftarrow$ firstNonFixatedReal($\gamma$)
7      foreach $tr_i \in tr$ do
8          $f(tr_i) \leftarrow f(tr_i) + 1$
9          foreach $r \in$ regretCustomers do
10             if Interchange of $r$ and $tr_i$ is possible then
11                 $f(r) \leftarrow f(r) + 1$
12             end
13         end
14     end
15 end
16 $\gamma^c \leftarrow \arg\max(\gamma \in \Gamma) \sum_{i=1}^{m} f($firstNonFixatedReal$(\gamma)_i)$
17 return firstNonFixatedReal($\gamma^c$)
```

Another option would be to calculate regret of visible real requests from the entire plan, that has not yet been served. This method will make the algorithm even less elitist. Defining γ_u to be the part of the plan that is not fixated, the set Re would be:

$$Re = \gamma_u \bigcap F \tag{4.2}$$

Although consensus has only been calculated for real requests so far, strategies for accepting sampled requests have been developed by Van Hentenryck and Bent with good results. These will be explained in the next section. In terms of regret, this means that it might also make sense to consider the regret of sampled customers. In the same way as above, this could be done for both the current route, and the entire plan. This yields the sets 4.3 and 4.4, respectively:

$$Re_s = \rho_u \tag{4.3}$$

$$Re_s = \gamma_u \tag{4.4}$$

The main problem with calculating regret only for the current route, is that this might not eliminate the elitism sufficiently, since the increase in evaluation due to regrets might be very small. Remember that the regret customers have to be interchangeable with the customer ts_i, which means time windows have to fit which is more unlikely the further ts_i and the regret customer are from each other in the plan, time wise. On the other hand, calculating regret for the entire remaining routing plan, might result give too influence to the regret-calculation.

Furthermore, calculating regret for all the customers in the plan is time consuming. This is examined imperically in section 9.2.

4.5 Two Alternative Strategies: Waiting and Relocation

The Consensus and Regret algorithms described above only consider real requests when deciding which customer to serve next. In Van Hentenryck and Bent [2006], two alternative strategies are suggested in which sampled customers are also considered. There are some inherent problems when only considering real customers. Imagine, for example, that there is consensus for some customer c at a given time t. It could be that c's time window is starting ($e(c)$) quite far from t, in which case the assignment of c to the route in question, would mean the vehicle will drive to c and simply wait. While $e(c)$ is likely to be reasonably close to t, the waiting time at c might be better spent. Furthermore, if there is a high probability of a customer materializing (ie. a customer that was sampled actually appears) in some area, this would ideally be expressed by a consensus on serving a sampled customer in this area by the pool of sampled plans. This information is lost when only considering real requests, and if the customer indeed materializes, it is likely that the final result would not have been as good, as if the consensus for the sampled customer had been taken into consideration.

The question of what to do if there is consensus of a sampled customer still remains. Van Hentenryck and Bent presents two strategies: Waiting and Relocation, explored in the following subsections.

4.5.1 The Waiting Strategy

The Waiting strategy is based on the recognition that in some circumstances, it might not be desirable to rush to the first real customer, and then just wait there, like described in the above example. Instead, if there is consensus for at sampled customer, the vehicle waits at its current position. In this way, new requests will have a chance to appear and be taken into consideration. This strategy is particularly useful for instances in which the bottleneck is the minimization of traveling times, and it is relatively easy to serve all the customers. For the problems of online stochastic vehicle routing similar to those treated in this thesis where serving all customers is hard, Van Hentenryck and Bent documents in their book, that the Relocation strategy yields better results. For this reason, the waiting strategy has not been implemented in connection to this thesis, but for details of the algorithm the reader is referred to Van Hentenryck and Bent [2006].

4.5.2 The Relocation Strategy

While the waiting strategy addresses the problem of serving the real requests too eagerly, the problem with simply waiting when a sampled customer is consensus, is that it is not particularly efficient when the objective is to serve as many customers as possible and this is hard. An alternative strategy, dubbed Relocation, still recognizes the potential advantage in not rushing to serve real customers, but does this by treating sampled and real customers equally, and serving the consensus customer, even if it is sampled.

If the consensus is a sampled customer, this means there is an increased probability for a real customer to appear at that location. By moving to the sampled customer, the vehicle will be closer, in case the customer appears. While this might seem drastic, Van Hentenryck and Bent have achieved good results with this strategy. Note that this strategy is never desirable when the primary objective is to minimize travel distances.

In Algorithm 10, the Relocation strategy is shown for the Consensus algorithm. The Regret algorithm can be modified in a similar way to implement Relocation. In line 1, the default evaluation is set to $f = 0$. Line 2-9 are essentially identical to the last part of the original Consensus, except that we now do not filter out sampled customers. We call a function `firstNonFixated`, rather than `firstNonFixatedReal`, and hereby take the first customers following the partial plan, disregarding whether they are real or sampled.

Algorithm 10: chooseRequest-$\mathcal{CR}$

```
Data: Sampled routing plans (Γ)
Result: List of consensus customers
1 Set default evaluation $f(c)$ to 0 for all $c$'s.
2 foreach $\gamma \in \Gamma$ do
3     $tr \leftarrow$ firstNonFixated($\gamma$)
4     for $i \in 1..m$ do //recall m is number of vehicles
5         $f(tr_i) \leftarrow f(tr_i) + 1$
6     end
7 end
8 $\gamma^c \leftarrow \arg\max(\gamma \in \Gamma) \sum_{i=1}^{m} f($firstNonFixated$(\gamma)_i)$
9 return firstNonFixated($\gamma^c$)
```

There are a few implementational considerations to be made in connection to the Relocation strategy. Van Hentenryck and Bent have not described their approach, so the approach taken in this thesis is based on the authors own considerations. One such consideration to make, is how to “service” a sampled customer. The one obvious solutions would be to either simulate servicing a real customer, by driving to it, wait until ready, stay for full service time, and then drive on to next customer, unless of course the customer is materialized in the meantime, in which case the vehicle immediately starts service. Another solution would be simply drive to the sampled b customer, and hope it materializes on the way. If this is not the case, the vehicle simply moves on, as soon as it arrives. Both approaches have weaknesses. In the first, much waiting is potentially involved in case the customer does not materialize, which was what was attempted avoided in the first place. For the second approach, the problem is that the vehicle might leave the customer before it is materialized.

For this thesis, the first approach was used, in which a simulation of service takes place at the sampled customer. If it materialized, service is started at the customer as soon as possible. This, of course, requires a specification of the criteria for materialization. This was not detailed by Van Hentenryck and Bent either, but the approach used in this thesis will be described in section 4.8.

4.6 Allowing Arbitrary Locations and Samples

To make the sampling realistic, it seems reasonable to allow customers to appear in all possible locations with all possible time windows, demands and service times. However, a problem arises if this is allowed, since the probability of sampling two customers at the exact same position becomes smaller when the number of potential sample locations increase, ie. it is reversely proportional to map size and sample resolution. This means that the probability of having consensus for a sampled customer is increasingly small (especially if the map is continuous in having infinitely many locations). One way to handle this is by discretization of the map. Doing this, one can define customers in the same discrete area, to be considered as samples representing the same customer. This will be described in detail in section 4.6.1, below.

Despite a small e-mail correspondence with the author Pascal Van Hentenryck, it is not clear how they handle this problem in the book, except that they do not use discretization of the map. Even in an article of Bent and Van Hentenryck [2005], dedicated to the subject of sampling in connection to the Consensus and Regret algorithms, it is not quite clear. It is worth mentioning that it is only when using the Wait and Relocation strategies that the problem arises, because in only these cases is consensus of sampled customers involved.

It might be a deliberate decision based on two facts. If the sample base is eg. 400 predefined potential customers (with predefined time windows etc), each with some probability of appearing, and the instance size is 100, this dramatically increases the probability of consensus of sampled customers. Having a sample base of this limited size makes the Wait and Relocation strategies reasonable. If this is indeed the sampling approach used by Van Hentenryck and Bent, defining two samples equal only when they have the exact same properties makes sense. Having a sample base of, say, 40.000 predefined customers on a 100 customer instance, makes these strategies less useful. Another sample base could be by having some distribution model for different areas of the map, defining distributions of locations, time windows, demands, etc.. This approach would make the need for a discrete map even greater.

In the end, it comes to down to the nature of the problem at hand. If one considers a problem in which customers will always appear on fixed point on the map, with sufficiently high probability to make consensus of samples influential, the approach of Van Hentenryck and Bent could be used. If, on the other hand, one considers a problem in which customers can appear anywhere on the map, and potentially with any time window, demand etc., a discretization of the map is needed.

In any case, Van Hentenryck and Bents description of their sample base is insufficient for reimplementing it without making assumptions for the details.

As described in the introduction, one of the aims of this thesis is to attempt to make the sampling mechanics more general and realistic, and modify the algorithms to handle this. As mentioned this can be achieved by discretization of the map.

4.6.1 Discretization of the Map

As described above, the idea is to discretisize the map, into small areas. Rather than utilizing the probability for some customer to emerge, one utilizes the probability of a customer appearing in a discrete area. This seems realistic for eg. routing police vehicles. Here there might be areas with particularly high crime rate, like bars, ghettos or high-class stores. It is highly unlikely that any single jewelry store could have a high enough probability of being robbed to create any kind of consensus. But if one looks at an area of, say, 15 high-class stores, there might be sufficiently high probability, to result in a potential consensus to send a car there.

When discretizing the map, it is necessary to consider the degree of discretization. The idea of discretization is to make a method for either identifying sampled customers as representing the same, or group an area in which it does not matter for the routing plan which customer is appearing (because they are so geographically close). If the discrete areas are too large, samples appearing in the area might not represent the same customer, or customers might be too far away from each other for it to make sense to group them. Having areas that are too small make the approach more like that of Van Hentenryck and Bent, resulting in a an undesirable low probability for sampled customers having consensus. The larger the discrete areas are, the greater the probability of two different sampled plans have a sampled customer in the same square, and consequently greater probability of selecting that area as consensus. In effect, this means the a more detailed discretization, results in a higher priority of real customers. The effect of discretization is tested and tuned in section 9.3.

When implementing the discretization some changes in the algorithms are needed. First of all, the evaluation $f(c)$ should now be done on the discrete areas, rather than on customer in the Consensus and Regret algorithms with Relocation. When a consensus has been found for a plan, the consensus is for areas rather than customers and so might be based on several customers - if relocation is used, these might include both sampled and real customers. A decision has to be made on which one of these customers to actually serve next. For this thesis, the customers from the routing plan γ with the best evaluation are selected. Another option, when using Relocation, would have been to prioritize real customers over sampled ones: so if the plan γ has a sampled customer c_s as next for a route, but the area of c_s contains a real customer c_r, c_r would be selected instead c_s as consensus. This might help the algorithms to be able to serve more (real) customers; but there are some complications. It needs to be checked whether the replacement of c_s with c_r is feasible for plan γ. First of all, the difference in time windows, location, service time, etc., might make the plan invalid. Secondly, a check of γ's routes must be done to ensure that c_r is not present elsewhere.

When a request has been chosen, and added to the partial route, pruning has to be made. This is due to the fact that even though two customers a and b, from plans A and B, respectively, are in the same discrete consensus square, only one, say a, is chosen. It might not be the case that plan B is compatible with a (replacing b with a is infeasible) due to time constraints, demand or something similar. In this case, the plan B has to be pruned from the pool of sampled plans, even though it agreed on the consensus. Pruning of plans is described more thoroughly in the following section.

4.7 Pruning of Sampled Plans

When a change has been made to the partial plan, such as adding consensus customers to the routes, the pool of sampled plans has to be pruned for incompatible plans. There are several ways this can be done, and it depends on whether the Relocation strategy has been used. Van

Hentenryck and Bent prune all sampled plans that do not have the exact same customers as next in their route as the partial plan. If the Relocation strategy is not used, this means that the first real request of the consensus route and sampled route should be the same. If this is the case, all sampled requests before this on the sampled route, should be removed, thus serving the real customers as the next. When it comes to the Relocation strategy, Van Hentenryck et al. do not specify ther criteria for two sampled customers being the same. But if one only has a limited sample base of say 400 possible customers, the equality of two sampled customers most likely means it is the same exact sample of the 400 available, and so it is likely to be handled in the same way as without the Relocation strategy.

The Consensus and Regret algorithms depend on the pool of sampled plans to make good decisions, and one risk with pruning in this manner, is that too many plans might be pruned, resulting in a very small pool. Furthermore, having a big or continuous sample base when it comes to sampled customers in the Relocation and Wait strategy becomes negligible. An alternative strategy, implemented for this thesis, is to attempt to modify the sampled plan to fit the customer of the partial plan.

Algorithm 11: Prune for Relocation

```
Data: Sampled routing plans ($\Gamma$), current partial plan ($\gamma_t$)
Result: Sampled routing plans ($\Gamma$)
1  foreach $\gamma \in \Gamma$ do
2      for $i \leftarrow 1$ to $m$ do
3          $c \leftarrow$ last request on vehicle $i$ in $\gamma_t$
4          $c_s \leftarrow$ firstNonFixated($\rho_i \in \gamma$)
5          if $c = c_s$ then
6              continue
7          else if sameArea($c, c_s$) $\wedge$ replace $c_s$ with $c$ in $\rho_i$ if feasible $\wedge$ $c_s$ is sampled
           then
8              insert $c$ in place of $c_s$ in route $\rho_i$ of plan $\gamma$
9          else
10             $\Gamma \leftarrow \Gamma \backslash \gamma$
11             break route loop
12         end
13     end
14 end
```

In Algorithm 11 this algorithm is outlined. In the outer loop (line 1-14), each sampled plan is examined. An inner loop (line 2-13) runs through each of the routes. First, the last request c of the route of the partial routing plan is found, and then the first non fixated customer c_s of the current plans route is found. If the two requests are the exact same, we simply continue to the next route (line 5-6). If not, we check whether they are in the same discrete area and if it is possible to insert customer c in the place of c_s. If this is the case, we do not have to prune the plan, because the plans will still be compatible, as long as c_s is replaced by c, which is done in line 8. There is one more criteria for this replacement. This is due to the fact that c_s could indeed be a sampled representation of c in which case it makes great sense to do the replacement. If c_s is real, the plan is pruned because in this case it does not meet the consensus requirements, since a real customer will never represent anything but itself.

In terms of the regular Regret and Consensus algorithm, ie. without Wait or Relocation, the pruning is done based on first real customer in the sampled route. Although this is not specifically described by Van Hentenryck and Bent [2006], this seems to be the most obvious way of implementing the pruning. So a plan is pruned if the first non-fixated real customer is not the same as the customer c from the partial plans route.

4.8 Materialization of Customers

When using the Relocation strategy the customer selected as consensus can be a sampled customer. If a sampled customer is consensus, from here on referred to as a waypoint when assigned to the partial route, it means that the consensus of the sampled plans is a sampled customer. In other words, statistically, there is a relatively large probability for a customer appearing at that spot, so the sampled plans agree on moving there.

The problem of having a sampled customer in the partial plan is that, since it is not real, the vehicle sent to serve it has no actual customer to serve. Rather it has to wait at the partial customer, until "service" is done and it is assigned a new customer to visit. In this time, the vehicle neglects to serve any real customers which seems undesirable. Since the sampling has shown a high probability for a customer appearing in that position and time span, the probability of a real customer appearing with similar properties, is relatively high. If one such materialization takes place, we need to allow for the vehicle to immediately serve this new customer and stop service at the sampled request currently being served.

In a real life problem this would be similar to having a vehicle, eg. police car, drive to some area where the sampled plans has indicated a high probability of service being needed, eg. an area of bars at closing time. While the vehicle stays in that area, "serving" a sampled customer, a real customer calls for service. The vehicle then immediately leaves for this new customer.

In practice, whenever a new customer becomes visible, it is checked whether this customer is a materialization of a current waypoint being served. This can include time window, demand and position. If these criteria fit, the waypoint immediately stops being "served", and the new customer is served as the next in the partial route. Next, the sampled plans are updated to include this new customer, and pruned if the inclusion if infeasible.

Algorithm 12 outlines the algorithm for materialization of a customer. The outer loop iterates through the routes. We are only interested in the routes which ends in a sampled customer (line 2-4). Line 5 checks whether some criteria for materialization between this and the new visible customer holds. If this is the case, the actual materialization takes place. In line 6-7, we set the earliest departure for c_l to the current time, allowing the vehicle to leave right away. We then add the materialization c to the route. Line 8-18 loops through all the sampled plans, attempting to insert c into the plan in the same manner as with the partial plan. Since the sampled plans, unlike the partial one, have customers following c_l, they do not necessarily have room for the materialization c, in which case they are pruned (line 12). In case a materialization has occurred, the function stops, having successfully materialized the new customer c. If no candidate is found

Algorithm 12: materialize

```
Data: New customer $c$, partial routing plan $\gamma$, current time $t$
Result: Materialization of $c$ if criteria is meet, pruning of infeasible sampled plans
for $i \in 1..m$ do
    $\rho \leftarrow$route $i$ of partial plan $\gamma$
    $c_l \leftarrow$ last fixated customer in $\rho$
    if $c_l$ is a sampled customer then
        if criteriaMet($c_l, c$) then
            $\delta(c_l) \leftarrow t$
            add $c$ to $\rho$
            foreach $\gamma_s \in \Gamma$ do
                $\rho_s \leftarrow$ route $i$ of $\gamma_s$
                $c_s \leftarrow$ last fixated customer in $\rho_s$
                if criteriaMet($c_s, c$) then
                    $\delta(c_s) \leftarrow t$
                    add $c$ to $\rho_s$
                else
                    $\Gamma$
                end
                $\leftarrow \Gamma \setminus \gamma_s$
            end
            break for loop
        end
    end
end
```

for materialization, the partial plan is left unchanged. Note that this function should be called for each new visible customer.

It is worth noticing that the function prioritizes the first route in which a materialization can occur, rather than looking through all possibilities, and materializing the customer that fits the newly visible customer the best. A way this could be achieved, is to define a criteria of fitness for a materialization, iterate through all routes, and materialize the customer with the best fit.

Left to define is the `criteriaMet` function. In terms of location, the sample and its materialization should be in the same square, as explained in section 4.6: Allowing Arbitrary Locations and Samples.

$$e(c) \leq \delta(c_l) \tag{4.5}$$

$$z(c) > \max(c_l^- + d(c_l^-, c_l)), t) + d(c_l, c) \tag{4.6}$$

In terms of time, two criteria for equality are defined in equation (4.5) and (4.6). Equation (4.5) accepts materialization only if it is ready to be served before the earliest departure of the sampled customer. (4.6) checks if we are able to reach the new customer, if we leave the sampled customer as soon as possible. Note here that c_l^- is the customer previous to c_l. If the vehicle is on the way to the customer, we allow it to reach the customer first to avoid the complications of having the vehicle change directions mid-route. Else we can simply leave it at current time t.

$$Q \geq q(\rho) - q(c_l) + q(c) \tag{4.7}$$

The last criteria to check is given in (4.7). It simply makes sure that the materialization of the sampled customer is not violating the capacity of the vehicle.

If these criteria hold, the earliest departure of the materialized customer can be calculated. The incompatible sampled plans are now pruned, so only valid plans are left, with the new materialized customer.

5 Instances

To be able to test the algorithms described in this thesis, two different categories of instances are used. For benchmarking the offline algorithms used as sub-procedures in the Online Stochastic Algorithms, the Solomon instances are used. For testing algorithms requiring stochastic knowledge the instances have to be generated, since no instances with a known probability distribution were available. The instances, along with functionality for converting them to online instances are described in this chapter.

To investigate the performance and quality of the offline algorithms, benchmark instances are needed. A very common set of benchmarks to use when dealing with the VRPTW are the Solomon instances [Solomon, 1987]. These will be described in section 5.1.

Several of the algorithms implemented in this thesis depend on the existence of stochastic knowledge of the instance they are used on. This is not available for the Solomon instances. Generation of instances with stochastic knowledge available is done by the use of an instance template, implemented for this thesis, and described in 5.2. The benchmarks generated with the use of this instance template are described in section 5.3.

The Solomon instances are offline instances, meaning that all requests are known initially. The instances generated by the template are also offline. To make the instances online, some customers need to appear during the day. To achieve this, the approach of Van Hentenryck et al. is used, in which 5 classes of hardness are defined. Offline instances are converted to online instances by defining at what time the requests will be made, ie. what time they become visible to the solving algorithm. The lateness of the visible times are dependent on which class the online algorithm belongs to. This is be described in section 5.4.

5.1 Solomon Benchmarks

To be able to compare the efficiency of the implemented algorithms to those of others, common benchmark instances are needed. The choice of the Solomon benchmarks [Solomon, 1987] is based on the fact that they are some of the most common benchmarks in VRPTW, and they challenge the algorithm in different aspects.

The Solomon instances make up a total of 56 instances and are split into 6 problem sets. The naming convention of the instances is `DTm`. `D` defines what class the instance is; R, C or RC. In R, the customers are positioned randomly, in C they are clustered, and in RC the positions are a mixture of these. `T` can be either 1 or 2, where instances of type 1 have a short scheduling horizon, allowing only few customers per route. Type 2 instances have a long scheduling horizon allowing many more customers being served in one route. Besides this, the instances differ in terms of their time windows. Overall this gives a testing environment that challenges the algorithms in many different aspects.

For each Solomon instance, versions with 25, 50, 75 and 100 customers exists, all on a 100 by 100 unit map. In the context of this thesis, only the instances containing 100 customers are considered.

The most recent (optimal) solutions that could be found for these instances were found in April 2008 [Jepsen et al., 2008]. For several of the instances, optimal solutions have not been found. Furthermore, to be able to compare the implemented algorithm with alternatives, it is relevant to look at heuristic solutions found. M.M. Solomon has compiled the best solutions found by heuristics [Solomon, 2005]. Although this was last updated March 2005, it has not been possible to find more recent heuristic solutions. A compilation of both optimal and heuristic solutions can be found in Table 6.3, page 53.

5.2 Instance Template

To be able to test the Online Stochastic Algorithms, a stochastic model or historic data needs to be available for the instances the algorithm is solving. When solving a real life instance, this could be accomplished by the use of historic knowledge, statistics or something similar. It has not been possible to acquire real life data within the time frame of this thesis, so this is not an option. Furthermore, only having a few real problems does not allow one to test the algorithm on a variety of instance types.

For testing the algorithms on different instance types with historic data available, these algorithms (and stochastic data) were generated by means of an instance template.

The basic idea is to split the map into smaller areas. For each of these, probability distributions are defined for customers demand, time windows, ready times, etc.. When this has been set up, the instance template can now be used to sample instances. If one wants to create an instance, this can be done by creating a template p. The main instance can then be generated by sampling p and defining this as the main instance. This instance can now be solved, and p can be used to sample instances with similar stochastic properties.

Below, the properties that needs to be defined for an instance template are explained.

General: Before looking at the specific areas, some general settings should be defined for the instance template.

The dimensions of the map, the time horizon, location of depot, vehicles available and vehicle capacity are all properties of the entire template, and not the specific areas. These are not given as probabilities but are the same in all the instances produced by the template. The reason for this is, that realistically these parameters do not change from day to day.

The number of total expected requests for a day can be defined as an interval $[a; b]$. Furthermore, probabilities for few, medium or many requests are defined. These are probabilities of a number being sampled in the first, second or third part of the interval, respectively.

Areas: The map is split into a number of areas, that are either empty, sparsely, medium or densely populated. These will remain the same in all generated instances, since, realistically, a heavily populated area almost always remain heavily populated. Each area of the map is defined as being in one of the above mentioned categories.

For an instance template, the probability that a customer is in a sparse, medium or dense area is specified. Obviously empty areas will have no customers.

For each area an individual setting of constraints is defined. These are explained below:

Ready Time: A constraint of ready times is defined for each area. There are four probabilities defined; the probability that a customer has a start time at 0, in the first, second, and third part of the total time horizon.

These ready times are sampled independently of customer location, and therefore it is necessary to check that the sampled ready time is no later than it is possible to serve the customer and get back to the depot before h. If this is not possible, the ready time is corrected as to allow this. For this reason, the probabilities are skewed a little, but this is done to avoid setting infeasible ready times.

Window Length: The window length of customers in the area is given by probabilities of a customer having a very short, short, medium, long or very long time window. The length of these time windows are defined as $[2; 9]$, $[10; 18]$, $[19; 35]$, $[36; 55]$ and $[56; 80]$ percent of the total time horizon, respectively.

If a window length extends beyond the instance time horizon, the window is simply cut off at h. This skews the actual probabilities of customer's window length, but was done to simplify the instance template.

Demand: For demand, the minimum a and maximum b allowed demand for a customer is defined. Furthermore, four probabilities are given; the probability that a customer has a demand of 0, a value in the first third, a value in the second third, and a value in the third third of the $[a; b]$ interval.

Service Time: Like demand, this constraint defines minimum and maximum service time. It also defines probability that a customers service time is in the first, second and third part of the interval, along with a probability that the service time of the customer is 0.

Other: It should be mentioned that the customers are sampled independently. Therefore, sampling one customer in a dense area for example, does not decrease the chance of another customer appearing in that area. Furthermore, in the intervals defined above, customers are evenly distributed. Eg. a customer who has a very short time window, will be assigned a window length anywhere in the interval 2% - 9% at a uniform distribution.

Each of the settings described above, have a `sample` function defined, sampling the property according to the probabilities defined. So when a customer is sampled; first it is found in which type of density he appears, and a random area of the given type is picked. Each property (ready time, window length, etc. described above) of the request is then sampled according to the settings for that specific area.

Setting	**Long**		**Short**	
Name	**600Loose**	**600Tight**	**180Loose**	**180Tight**
Time Horizon (h)	600 *(3 CPU-sec/timestep)*		180 *(10 CPU-sec/timestep)*	
Customers	50 ± 5		50 ± 5	
Vehicles	5	8	12	14
Window Length	2-55% *(0.1/0.35/0.35/0.2/0)*	2-9% *(1.00/0/0/0/0)*	2-55% *(0.1/0.35/0.35/0.2/0)*	2-9% *(1.00/0/0/0/0)*
Ready Times	*(0.4/0.3/0.2)*	*(0.4/0.3/0.2)*	*(0.3/0.3/0.15)*	*(0.3/0.3/0.15)*
Service Time	5-25 *(0.2/0.6/0.2)*	5-25 *(0.1/0.4/0.5)*	5-15 *(0.2/0.6/0.2)*	5-15 *(0.2/0.3/0.5)*
Demand	2-15 *(0.2/0.7/0.1)*	2-15 *(0.2/0.6/0.2)*	25-55 *(0.3/0.6/0.1)*	25-55 *(0.2/0.6/0.2)*

Table 5.1: Properties of the Generated Benchmarks: Each benchmark has 50 customers, but when sampling from template these range from 45-55. For distributions within the window length, the reader is referred to the description in section 5.2 above. For ready times, *(a/b/c)* means that $a \cdot 100$ percent of customers has their ready time in first third of the time horizon. *b* and *c* denotes the second and third part of the horizon. For service time and demand, the interval given is the range of values the customers can take. The *(a/b/c)*-part denotes distribution, where a is the fraction that has a value in the first third, b the second third, etc..

5.3 Generated Benchmarks

The instance template framework described above, was used to create some benchmark instances for the Online Stochastic Algorithms. Obviously, the Solomon benchmarks are not useable for this, since no stochastic data are available for the instances. A total of four benchmark instances have been created, and like the Solomon instances, these are attempted made in a way so they challenge different aspects of the solving heuristics.

The real life time horizon for the Online Stochastic Algorithms described in this thesis would usually be very long. For example, it could run on an eight hours working day, generating and solving instances when time is available. This is not realistic when testing the algorithm, due to the enormous amount of time this would require[1]. Instead a real life time horizon of 30 CPU-minutes will be used. As an attempt to capture the properties of having a longer run time, the time consuming parts of the instances are also scaled down; number of customers, distances, and time horizon.

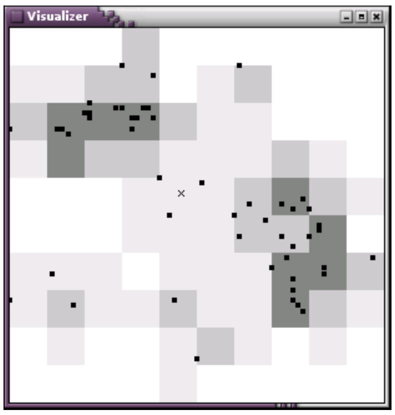
All the instances have 50 customers on a 70 by 70 grid. This means that approximately 1% of the squares are occupied by customers, like the Solomon instances. The capacity of vehicles are 200. Table 5.1 sums up the properties of the 4 instances. The two templates with a time horizon of 600 were constructed to generate instances with long routes (appr. 15 customers per route), whereas the templates with a time horizon of 180 only allows short routes (appr. 5 customers in each). For each type of time horizon, one template has very tight time windows (2-9% of time horizon) and one has long time windows (2-55%). Hopefully this allows examining how well the algorithms performs under different circumstances.

The number of vehicles available for each instance was chosen by solving the instance offline,

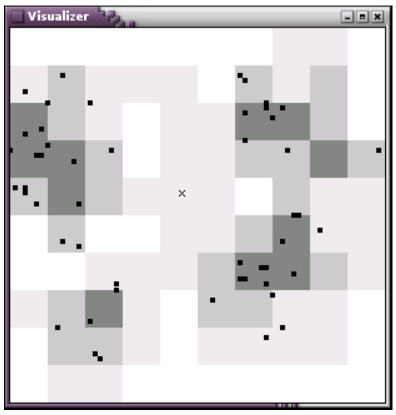
[1] A flawless testing of 5 settings on 5 instances would, with a time horizon of 8 hours, take 200 CPU-hours.

while minimizing the number of vehicles employed. The resulting number plus 1 vehicle was made available for the online solvers.

As described in section 5.2, the map is separated into fields, in which probabilities are specified for customer properties. In the generated instances, all the areas have the same setting. Although the templates allow different areas to have different distributions of customer setting, properly testing the influence of this is not feasible within the time frame of this thesis, so all the areas have the same customer properties. In terms of probabilities of where on the map customer appears, this is specified by the density of each area. Figure 5.1 shows the maps for the template[2] with a time horizon of 600 and 180. The darkest areas are dense areas, medium dark are of normal density, light are sparsely populated areas, and white areas are empty. In both templates, customers have a 50% chance of appearing in a dense area, 30% for a normal area, and 20% for a sparse. This statistically gives an average of 2.8 customers per dense area, 0.83 per normal and 0.29 per sparse area for the 600 instances. For the 180 instances, these numbers are 2.27, 0.714 and 0.27. Within the density categories, the exact area at which the customer appear is chosen at random. The customers of the actual benchmarks are shown as dots on the maps. Note that the customer locations for both the 600 instances are the same, this also goes for the 180 instance. Note that the instances are clustered, but with some random customers appearing, making this somewhat similar to the RC class of the Solomon instances. This choice was made to challenge the algorithm in both clustered and randomly positioned customers.



(a) Map of instance template and benchmark instance, for a time horizon of 600



(b) Map of instance template and benchmark instance, for a time horizon of 180

Figure 5.1: Maps of the Instance Templates and Generated Benchmark Instances: Darkest areas are densely populated, dark gray are of normal population, and light gray are sparsely populated. White areas are empty.

[2] A small GUI program able to display the instance template was created for this thesis. The screen shots from Figure 5.1 are of that GUI. As can be seen, it is possible to plot actual (multiple) instances on the map. Furthermore, the GUI has the ability to display multiple routing plans on the map, distinctive by individual colorings.

	T_1	T_2		T_3		
	H_0	H_0	H_1	H_0	H_1	H_2
Class 1	1.00	0.50	0.50	0.50	0.40	0.10
Class 2	1.00	0.50	0.50	0.50	0.10	0.40
Class 3	1.00	0.50	0.50	0.50	0.25	0.25
Class 4	1.00	0.50	0.50	0.20	0.20	0.60
Class 5	1.00	0.10	0.90	0.10	0.10	0.80

Table 5.2: Overview of Online Classes: Classes of distributions for visibility times of customers in the dynamic instances.

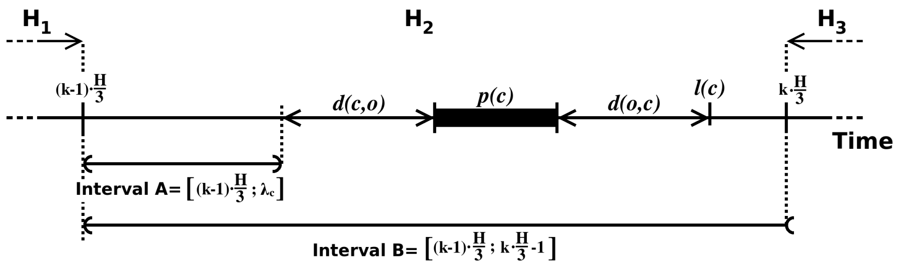

H_1
H_2
H_3
$(k-1)\cdot\frac{H}{3}$
$d(c,o)$
$p(c)$
$d(o,c)$
$l(c)$
$k\cdot\frac{H}{3}$
Time
Interval A= $\left[(k-1)\cdot\frac{H}{3} ; \lambda_c\right]$
Interval B= $\left[(k-1)\cdot\frac{H}{3} ; k\cdot\frac{H}{3}-1\right]$

Figure 5.2: Illustration of Online Classes: The figure shows the possible intervals for a customer c to be visible in the interval $k = 2$. He becomes visible at some point in time, uniformly drawn from the smallest of interval A and B.

5.4 Online Instances

The Solomon instances are all offline instances and so are the instances generated by the template. To create online versions of them, the same approach as Van Hentenryck and Bent was used, as described in Van Hentenryck and Bent [2006] and Bent and Van Hentenryck [2004a]. The basic data of the offline instance, such as time windows, service time and location is preserved. The instances only need to be extended in terms of assigning a feasible time to each customer at which he becomes available (ie. visible).

First the time horizon h is split into three periods of equal size; H_1, H_2, and H_3. Furthermore, we define H_0 to represent the time before the day starts (meaning that customers belonging to H_0 are visible from the beginning of the day). Each customer is then assigned to one of three types (T_1, T_2 or T_3), according to their due time and distance to the depot. More specifically, a vehicle has to be able to finish servicing a customer at his due time, and travel back to the depot, before the period is over. So a customer c is of type T_2 if and only if $l(c) + p(c) + d(c, o) \in H_2$, where o is the depot.

A customer of type T_1 is visible in H_0 (beginning of the day). Type T_2 customers become visible in either H_0 or H_1, while T_3 customers become visible in H_0, H_1 or H_2: which period is defined by some distribution depending on the online class (see Table 5.2). No customers are assigned to time period H_3.

To be able to test their algorithm on varying degree of dynamics, Van Hentenryck and Bent created 5 classes of distributions, seen in Table 5.2. The class defines for each type T_x, the probability of which period a customer becomes available. In an instance of Class 5, for example, a customer of type T_1 will always be visible from the start of the day. A type T_2 customer will have a probability of 10% to become visible at H_0 and 90% chance customers to become visible at some point during H_1. Finally a T_3 customer will become visible in H_0 with 10% probability, in H_1 with 10% probability or in H_2 with 80% probability.

Note that the later customers becomes visible the harder the instance class is. Also note that all customers are visible at time H_3.

The time in which a customer becomes visible during some interval $H_k, k \in 1, 2, 3$ is drawn uniformly from the interval:

$$[(k-1) \cdot \frac{H}{3}, \min(\lambda_c, k \cdot \frac{H}{3} - 1)] \tag{5.1}$$

Where $\lambda_c = l(c) - (d(c,o) + p(c) + d(o,c))$ is the time it takes a vehicle to travel to customer c from depot o, service him, and return to the depot. An example of this is illustrated in Figure 5.2.

6 Offline Algorithms

In the Online Stochastic Algorithms described in section 4, the stochastic VRPTW is repeatedly transformed to an offline instance and solved. This is done by sampling the unknown part of the instance, which has the effect that regular offline algorithms, made for VRPTW become applicable. Only minor changes needs to be done in the parts of the routing plan they are allowed to change: this was described in Chapter 4.

In this chapter, algorithms needed for solving the offline VRPTW will be explored. In section 6.1, algorithms for constructing solutions will be described. A brief examination of neighbourhoods for the VRPTW is done in section 6.2. Finally, two meta heuristics are explored in section 6.3 and 6.4 along with a comparison of these, which can be found in section 6.5.

6.1 Construction Heuristics

Route construction heuristics are used to create an initial routing plan from a set of unrouted customers in an instance. On a well studied problem like the VRPTW, there has been developed many construction heuristics, amongst the best known is the savings heuristic by Clarke and Wright [1964].

Although several objective functions are used in the context of this thesis, only one construction heuristic has been implemented. The objective of minimizing length (w_0, see equation (2.6), page 8), is only used to compare when testing the efficiency of the Attribute Based Hill Climber (see section 6.3). But preliminary runs, with the implemented construction heuristic yielded very good results, and it was found not to be necessary to implement another construction heuristic for this purpose only.

The other two relevant objective functions are minimization of vehicles used, and minimization of unserved customers, with minimization of route length being a secondary objective in both cases. The chosen construction heuristic is a sequential method, in that it builds the routing plan route by route, as opposed to constructing multiple routes simultaneously. This means it fills out a route before starting the next. This is suitable for both objectives, and will be explained in the following.

6.1.1 Impact

Braysy and Gendreau [2005a,b] presents a survey of the research done on non-optimal algorithms for the VRPTW. In Braysy and Gendreau [2005a] route construction and local search algorithms for VRPTW are examined. Here, a comparison of different route construction heuristics is carried out, and while the results are not unambiguous in terms of which is the best algorithm, the `Impact` construction algorithm of Ioannou et al. [2001], seems to perform well in general.

The `Impact` algorithm minimizes the number of vehicles and secondly the route length. It is sequential, and based on the insertion heuristic of Solomon [1987]. First a new route is initialized with a seed customer. Based on some criteria, a new customer is selected and inserted into this route. This continues until it is not possible to insert any more of the unrouted customers into the route. Then a new seed customer is selected to initialize a new route, and customers are then added to this until no more insertions are feasible. This continues until all the customers have been added to a route, at which point the algorithm is done. In the `Impact` algorithm, the criteria for choosing a new customer to insert and the position in which to insert it, is based on the greedy look-ahead approach of Atkinson [1994]. In the terminology of Ioannou et al. [2001], this criteria is called impact, and is an attempt to calculate the impact that the insertion of a customer has on the customers yet to be assigned, while of course taking route length into account. Before further specifying the calculation of the impact criteria, the main algorithm, given in Algorithm 13, will be explained.

Algorithm 13: Impact

```
Data: Unrouted Customers
Result: Routing plan
1  U ← unrouted customers
2  while U ∉ ∅ do
3      $c_{seed}$ ← the customer ∈ U farthest from depot
4      $\rho$ ← initialize with $c_{seed}$
5      U ← U \ $c_{seed}$
6      repeat
7          best ← ∞
8          foreach $u$ ∈ U do
9              feasibles ← feasible positions for inserting $u \in \rho$
10             foreach f ∈ feasibles do
11                 imp ← calculate impact of inserting $u$ at the position f
12                 if imp < best then
13                     best ← impact value, customer and position
14                 end
15             end
16         end
17         make insertion as given by best
18         U ← U \ best
19     until best = ∞
20     add $\rho$ to the routing plan
21 end
22 return the routingPlan
```

We start with a set of unrouted customers (line 1), and continue the algorithm until the set is empty (line 2), meaning that we have inserted all the customers in routes. A new route is initialized with a seed customer, which is taken to be be the customer farthest away from the depot. After the initialization of the route, the “impact” of inserting each of the unrouted customers in each of the feasible positions in the route is calculated (line 8-16), and the best of these is inserted into the route (line 17). When no more customers can be inserted into the current route (resulting in line 19), it is added to the routing plan, and a new seed customer is selected to initialize a new route. The result is a feasible routing plan, in which all the customers are assigned. When the objective is to minimize the number of unserved customers, it might not be possible to assign all of them. In this case, the algorithm simply builds routes while vehicles are available, and then returns a plan of the constructed routes and the list of unserved customers.

We still have left to look at the core of the algorithm, namely the calculation of the impact the insertion of a customer has. The main idea is to minimize the effect the insertion of a customer u has on the customers already in the route and the unassigned customers while keeping route length short. Minimizing this effect, or impact, intuitively seems beneficial, allowing good insertions of the remaining customers. The main function is comprised of three parts, which we will examine, before looking at the final function.

The first part is given in equation (6.1), where $a(c_u)$ specifies the arrival time of the vehicle to customer c_u. Recall that $e(c_u)$ was defined to be the earliest start of service for customer u.

$$IS(c_u) = a(c_u) - e(c_u) \tag{6.1}$$

$IS(c_u)$ is an attempt to model the slack surrounding customer c_u after being inserted. In other words, this equation is meant to capture how much freedom there is for insertion of a customer before and after c_u. A non-negative value close to zero means more slack for the insertion of customers before and after c_u consequently giving more room for insertions.

The second part is somewhat related to (6.1), as it also attempts to capture the impact the insertion of the customer u has on the unrouted customers, and their future insertions. To avoid confusion, we here stick to the terminology of Ioannou et al. [2001], and define J to be the set of unrouted customers. After inserting a customer c_u, a necessary condition for the vehicle to visit some other customer c_j is $e(c_u)+p(c_u)+d(c_u,c_j) \leq l(c_j) \bigvee e(c_j)+p(c_j)+d(c_j,c_u) \leq l(c_u)$[1]. Selecting a customer that minimizes the non-negative difference of $[l(c_j) - \big(e(c_u) + d(c_u,c_j) + p(c_u)\big)]$ or $[l(c_u) - \big(e(c_j) + d(c_j,c_u) + p(c_u)\big)]$ for all $c_j \in J$, is expected to be a good selection. Which of these is positive obviously depends on the order of c_j and c_u. This leads to the criteria:

$$IU(c_u) = \sum_{c_j \in J - c_u} \frac{\max\big(l(c_j) - e(c_u) - d(c_j,c_u) - p(c_u), l(c_u) - e(c_j) - d(c_u,c_j) - p(c_j)\big)}{(|J| - 1)} \tag{6.2}$$

The last criteria is called internal impact, and considers the effect the insertion of customer c_u has on the route, when inserting it between customer c_i and c_j. This criteria is comprised of

[1]Here, and in equation (6.2), the implementation of this thesis differs from that of Ioannou et al. [2001]. In their article, they write:

> A necessary condition for a vehicle to visit customer j after the selected for insertion customer u is (assuming feasibility): $e_u + d_{uj} \leq l_j \bigvee e_j + d_{ju} \leq l_u$

. In this formula they do not take the service time of the customer into consideration. The same is the case from their variant of the IU (equation (6.2)). Although they write "assuming feasibility", which ensures their statement is not incorrect, it seems to make more sense to also consider the service time in the expression and in the calculation of IU as well.

three parts. The first simply calculates the increase in route length:

$$ir_1(c_u, c_i, c_j) = d(c_i, c_u) + d(c_u, c_j) - d(c_i, c_j) \tag{6.3}$$

The next part calculates the difference in arrival time at customer j before and after the insertion of u — that is, the delay of arrival at j. This is done by:

$$ir_2(c_u, c_i, c_j) = [l(c_j) - (a(c_i) + p(c_i) + d(c_i, c_j))] - [l(c_j) - (a(c_u) + p(c_u) + d(c_u, c_j))] \tag{6.4}$$

The final part of the third criteria models the time gap between the latest service time $l(c_u)$ of customer u and the arrival time of the vehicle. This can be calculated as:

$$ir_3(c_u, c_i, c_j) = l(c_u) - (a(c_i) + p(c_i) + d(c_i, c_u)) \tag{6.5}$$

Putting all these together, weighted, we get an expression of what Ioannou et al. [2001] calls *local disturbance*. This has to be calculated for all feasible insertions of u and can be described by the equation:

$$IR(c_u) = \sum_{(c_i,c_j)\in I_r} \frac{b_1 ir_1(c_u, c_i, c_j) + b_2 ir_2(c_u, c_i, c_j) + b_3 ir_3(c_u, c_i, c_j)}{|I_r|} \tag{6.6}$$

Where $b_1 + b_2 + b_3 = 1$, $b_1, b_2, b_3 \geq 0$, and I_r is the set of all feasible insertion points of customer c_u.

Putting all this together, gives us a formula for calculating the impact of inserting a customer u into a routing plan:

$$Impact(c_u) = b_s IS(c_u) + b_e IU(c_u) + b_r IR(c_u) \tag{6.7}$$

Where $b_s + b_e + b_r = 1$, and $b_s, b_e, b_r \geq 0$.

The only thing left to discuss is the weights b_1, b_2, b_3, b_s, b_e, and b_r. A thorough testing of these in Ioannou et al. [2001], shows that the changes to the parameters from equation (6.6) shows statistically insignificant differences. It has therefore been decided to let these contribute equally, setting $b_1 = b_2 = b_3 = \frac{1}{3}$. The best setting of the other three parameters from (6.6) were not as unambiguous, and depended on the size of time windows. Ioannou et al. [2001] did the parameter tuning on the Solomon instances, and the only inference that they could make, was that for instances with small time windows b_r should be set greater than b_s and b_e to achieve good results.

Tuning of Parameters

In the context of this thesis, the construction heuristic might often run on sampled instances in which it is impossible to serve all customers. The parameters were tuned for minimization of used vehicles. To examine whether a parameter setting was suitable, and stable, for both cases, ie. sufficient and insufficient vehicles available, a race was setup. The values tested are given in Table 6.1. The values given in each line, are tested in all combinations on parameters. So for the

a	b	c
0.00	0.00	1.00
0.10	0.10	0.80
0.20	0.20	0.60
0.30	0.30	0.40
0.40	0.40	0.20
0.33	0.33	0.33
0.50	0.35	0.15

Table 6.1: Parameter Values for Testing the Impact Heuristic: For each line, except the last two, three parameters settings are tested. For the variables (b_r, b_s, b_e) the settings: (a, b, c), (c, a, b), (b, c, a) are used. In the second to last row, parameters are equal, so this setting is tested directly. For the last row, all combinations of the three values are tested with the given values.

.

first line, for example, the settings ($b_r = 0, b_s = 0, b_e = 1$), ($b_r = 1, b_s = 0, b_e = 0$) and ($b_r = 1, b_s = 0, b_e = 0$) are tested. The parameters was tested on the Solomon benchmarks. To capture the challenge of minimizing the unassigned customers, a run of the Attribute Based Hill Climber (described in section 6.3) was done on the instance, with the objective to minimize the number of vehicles employed. The resulting vehicles found, were set as the number available for the instance. This resulted in a set of instances, in which a preliminary testing with `Impact` showed it was able to assign all the customers in around half of the instances. Hereby `Impact`s ability to solve both instances where an sufficient and insufficient amount of vehicles are available were tested.

The race was done via the *race* package for the R statistics program[2]. The race was done unreplicated, meaning one run per setting on each instance. For elimination, the Friedman test was used with a 95% confidence interval. The output of the race can be found in Appendix B.1.

The only parameter setting that could be eliminated was the ones from the first row of Table 6.1, in which two of the parameters had no weight, ie. influence on solution. Other than that, no significant difference was found in the quality of the solutions, and so the choice was based on the conclusion of Ioannou et al., that having an increased value of b_r was desirable. The parameters for `Impact` is set $b_r = 0.40$, $b_s = 0.30$, and $b_e = 0.30$ for use in this thesis.

6.1.2 Ejection Chains

Although the `Impact` algorithm performed well, in comparison to the other construction heuristics examined in Braysy and Gendreau [2005a], an average of around 6 customers were not assigned to routes (according to the race described above). To further improve the starting solution, the use of Ejection Chains were examined.

The idea of Ejection Chains is to fit the unrouted customers into the plan, by continuously exchanging them with routed customers until room becomes available for an insertion.

More specifically, the Ejection Chain algorithm selects an unrouted customer c_u, and attempts

[2]The R-Project: http://www.r-project.org/

Instance	**Impact**		**Ej.Chain**		**Instance**	**Impact**		**Ej.Chain**	
	u	length	u	length		u	length	u	length
C101	7	1245.39	6	1303.47	**R112**	19	1574.12	1	1402.74
C102	1	2073.18	1	1990.56	**R201**	0	2490.93	0	2490.93
C103	1	2536.32	0	2299.31	**R202**	0	2147.67	0	2147.67
C104	0	1978.38	0	1978.38	**R203**	0	2105.30	0	2105.30
C105	10	1774.93	7	1701.27	**R204**	0	1864.73	0	1864.73
C106	13	2153.74	5	1903.67	**R205**	0	2295.75	0	2295.75
C107	6	1878.72	3	1901.96	**R206**	0	2334.61	0	2334.61
C108	9	2423.92	3	1714.50	**R207**	7	2054.32	0	1937.67
C109	2	1923.78	0	1936.40	**R208**	0	1912.13	0	1912.13
C201	0	1297.03	0	1297.03	**R209**	0	2506.80	0	2506.80
C202	11	1971.08	4	1381.11	**R210**	0	2667.28	0	2667.28
C203	0	2259.68	0	2259.68	**R211**	0	2385.57	0	2385.57
C204	12	2017.02	6	1557.39	**RC101**	4	2079.34	3	2056.83
C205	3	1088.45	1	1013.79	**RC102**	2	2031.08	1	2055.36
C206	7	1551.94	0	945.44	**RC103**	9	1820.69	3	1727.08
C207	8	1706.94	1	1164.46	**RC104**	34	1679.41	10	1430.22
C208	3	1195.44	2	1169.78	**RC105**	11	2232.00	3	1999.04
R101	1	2004.22	1	1959.64	**RC106**	7	1872.85	2	1732.95
R102	1	1967.22	1	1946.28	**RC107**	17	1835.44	5	1651.51
R103	0	1797.63	0	1797.63	**RC108**	24	1762.99	4	1566.82
R104	19	1612.33	4	1456.85	**RC201**	0	2498.03	0	2498.03
R105	6	1853.62	0	1734.82	**RC202**	0	2478.15	0	2478.15
R106	4	1882.32	1	1800.63	**RC203**	0	2554.39	0	2554.39
R107	17	1594.06	4	1441.31	**RC204**	0	2345.47	0	2345.47
R108	22	1478.28	8	1327.77	**RC205**	0	2750.94	0	2750.94
R109	5	1772.27	2	1670.24	**RC206**	1	2597.89	0	2608.50
R110	11	1679.07	5	1533.38	**RC207**	0	3077.34	0	3077.34
R111	17	1582.34	5	1445.10	**RC208**	2	2755.04	0	2685.74

Table 6.2: Comparison of Running Impact With and Without Ejection Chains: The instances used are Solomon instances with limited vehicles. In the headings, u indicates number of unserved customers, *length* is route length. For each instance, the left column, named *impact* is the impact algorithm with no ejection chain, whereas *Ej.Chain.* are a run of impact followed by the ejection chain algorithm.

to insert this into the routing plan in a best fit manner - the best insertion is selected. If insertion is possible, the customer is inserted and a new choice of unrouted customer is made. If no insertion is possible, the algorithm searches for a routed customer, c_i, that it can eject from a route and replace with c_u. The replacement yielding the best objective function is chosen. The customer c_i is now unrouted, and a new iteration starts. First, c_i is attempted inserted, and if this is not possible, the algorithm searches for a customer to replace (eject). Ideally these chains of ejections will make room for the insertion of the unrouted customers. The algorithm finishes when no more insertions or injections are possible.

A problem with the algorithm, is that the ejection chain might enter a loop, ejecting the same chain of customers. This can be remedied by a tabu list that dictates which customers are not allowed to be ejected. When a customer is inserted into the route, this should be added to the tabu list.

A comparison of runs of Impact only, and Impact followed by an Ejection Chain is given in Table 6.2. Looking at the table, it is clear that the Ejection Chain heuristic improves the solution significantly. For almost all routing plans where unserved customers are present, the algorithm improves the plan.

6.2 Neighbourhoods

Like previously mentioned, for a well studied problem like VRP, a great amount of algorithms have been created. This is also true for neighbourhoods in the VRP. When we look at the VRP with time windows, the amount is limited somewhat, because the time windows do not allow a part of a route to be reversed in terms of the ordering of the customers. As an example of one such neighbourhood, Figure 6.1 shows the `2-opt exchange operator` [Braysy and Gendreau, 2005a]. On the right figure, the dotted route has been reversed due to the neighbourhood move. This means that customers like $i + 1$, $i + 2$, etc, that used to appear early in route (and therefore are likely to have early time windows) will suddenly appear late in the route. This is potentially conflicting with their time windows. Obviously the same problem arises with customers j, $j - 1$, etc, who are likely to have late time windows. After the move, these appear early in the route, which might conflicted with their time windows.

Besides the comparison of construction heuristics, Braysy and Gendreau [2005a] contains a summary of popular neighbourhoods for the VRPTW. From these, two was chosen to be implemented in this thesis, namely the `relocate` and `exchange` neighbourhood. One reason for this choice was the simplicity of these neighbourhoods. The focus of this thesis is on the online stochastic algorithms, so two simple neighbourhoods seemed suitable. In the next two sections, these will be described in detail.

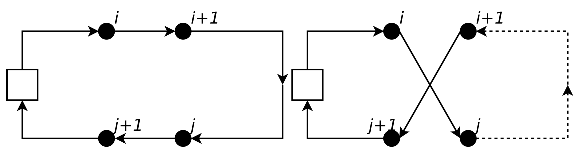


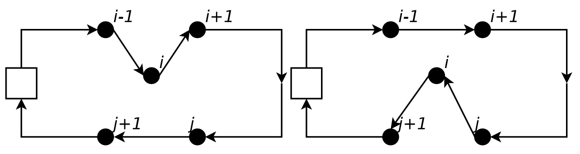
Figure 6.1: Example of Potentially Bad Neighbourhood When Using Time Windows: Figure showing the `2-opt exchange operator`. In the right figure, the direction of the route between i and $j - 1$ have been reversed (dotted line). This means that customers that appeared early in the route before neighbourhood move, will now appear late and vice versa. This is potentially conflicting with time windows.

6.2.1 Relocate

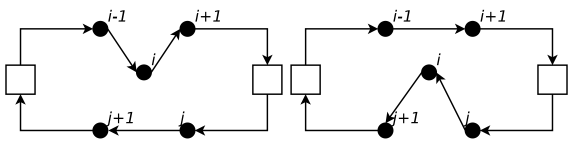
The principle of the `relocate` operator is to remove one customer from a route and inserting it somewhere else, either in the same route, or in some other route. The move is depicted in Figure 6.2. As can be seen, the way to move customer i from one position to another is to remove edges $(i-1,i)$, $(i,i+1)$ and $(j,j+1)$ and inserting the edges $(i-1,i+1)$, (j,i) and $(i,j+1)$.

As is the case of all neighbourhood moves in the VRPTW, `relocate` suffers under the stricter ordering on customers due to time windows. But there is a major difference between the impact of this ordering on the `2-opt exchange operator`, described above, and `relocate`. In `relocate` all customers on the relevant routes, not counting i, will only be affected by a shift in the time due to the removal or insertion of i. This shift, of course, is not irrelevant, and it is necessary to check if the time windows of the customers are violated. This is notably different than the `2-opt exchange operator` neighbourhood in which entire parts of the route is reversed, causing great change in visiting times for all the customers involved, and hence making most neighbourhood moves infeasible.

When using the objective function given in equation (2.9), where only a limited number of vehicles are available, we have a pool of unserved customers. This is handled in a straightforward way in the relocate neighbourhood. A relocate involving the unserved pool happens in two scenarios. A customer can be removed from a route, and inserted into the pool of unserved customers. Unlike a relocate involving two routes, there is no need to check time windows in this case, since the unrouted customer are not served at all, and therefore are by definition not served in their time windows in any case. The other scenario, is taking a customer from the set of unserved customers and inserting it into a route. In this case, the time windows of course needs to be checked.



(a) `Relocate` where customer i is inserted into the same route from which it is removed



(b) `Relocate` involving multiple tours. Here i is removed from one route and inserted into another.

Figure 6.2: Figure of the Relocate Neighbourhood Move: Shows the relocate neighbourhood involving a single or multiple routes.

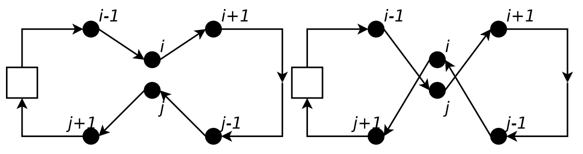
6.2.2 Exchange

The second implemented neighbourhood is called `Exchange`, and as the name suggests, the basic principle is to take two customers and exchange them. This is depicted in Figure 6.3, in which customers i and j are exchanged. This is done by replacing the edges $(i-1, i), (i, i+1), (j-1, j)$ and $(j, j+1)$ with $(i-1, j), (j, i+1), (j-1, i)$ and $(i, j+1)$.

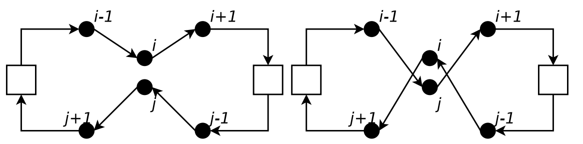
The exchange neighbourhood might not intuitively seem very usable, given the strict ordering VRPTW has. When having chosen a customer i, the selection of j is very limited, since the timespan between $j-1$ and $j+1$ has to match the time window of i, and furthermore allow i's service time $p(i)$. In the same way, the timespan between $i-1$ and $i+1$ has to allow for the service time of j and fit j's time window. In this way, the size of this neighbourhood is limited greatly by the time windows.

Unlike the case of `relocate`, the size ($|cust(\rho)|$) of the routes will also remain the same despite the number of exchanges done. This greatly decreases the number of solutions reachable by the sole use of `exchange`. On the other hand, `exchange` allows for some moves that might not be possible using the relocate neighbourhood exclusively. Therefore, using two neighbourhoods in conjunction might allow for some good solutions, whereas an exclusive use of exchange is not expected to be able to reach very good solutions, since the number of customers in the routes will remain the same, as will the number of unserviced customers.

As with `relocate`, the set of unserved customers needs to be taken into account when using the objective (2.9), page 9. Like `relocate`, this is handled by treating the unserved set as a route, and simply allowing exchanges between a route and the unserved set. The only difference is that the time windows do not have to be checked in the unserved set, since these sampled customers are obviously not being served at all.



(a) `Exchange` where customer i and j from the same route are exchanged. Note that i should be preceding j in the route.



(b) `Exchange` involving multiple tours. Here i and j, each from a different route, are exchanged.

Figure 6.3: Figure of the Exchange Neighbourhood Move: Figures showing the exchange neighbourhood for a single and multiple routes. Customers i and j are exchanged, resulting in i being served between $j-1$ and $j+1$ and j being served between $i-1$ and $i+1$.

Algorithm 14: Basic ABHC algorithm for VRP

```
Data: Routing Plan $\gamma$
Result: Routing Plan $\gamma_r$
$\gamma_r \leftarrow \gamma$
$\gamma_b \leftarrow \gamma$
while $\gamma_b \neq null$ do
    IdentifyWorstAttributes($\gamma$)
    $\gamma_b \leftarrow$ null (set w($\gamma_b$) to $\infty$)
    foreach $\gamma_t \in \mathcal{N}(\gamma)$ do
        if w($\gamma_t$) $<$ w($\gamma_b$) $\wedge$ Accept($\gamma, \gamma_t$) then
            $\gamma_b \leftarrow \gamma_t$
        end
    end
    if $\gamma_b \neq null$ then
        $\gamma \leftarrow \gamma_b$
        UpdateAttributes($\gamma_b$)
        if $\gamma_b < \gamma_r$ then
            $\gamma_r \leftarrow \gamma_b$
        end
    end
end
```

6.3 The Attribute Based Hill Climber (ABHC)

The Attribute Based Hill Climber algorithm (ABHC), is a relatively new algorithm, first introduced by Whittley and Smith [2004]. They applied it to the Quadratic Assignment Problem and Travelling Salesman Problem, for which it was shown to be competitive with existing algorithms. In 2006, Derigs and Kaiser applied the heuristic to the vehicle routing problem, described in Derigs and Kaiser [2007]. Here it was shown to be competitive with the best known heuristics for VRP. This has been further supported by the masters thesis of Nikolajsen [2009], in which it was applied to the Site Dependent VRP with Time Windows (SDVRPTW) and yielded very good results.

6.3.1 The ABHC Algorithm

The ABHC heuristic is a parameter free algorithm, based on principles from Tabu Search. In short, for a given type of problem, a set of atomic attributes (ie. edges in VRP) are chosen, for which the objective value of the best solution they have been a part of is associated. The algorithm searches through all the neighbouring solutions, picking the one with the lowest objective function. A neighbouring move is acceptable if the new objective value improves on one or more of the attributes that comprise the new solution (ie. routing plan). This continues until no more acceptable moves are possible, in which case the best solution found during execution is returned. Initially all the attributes are initialized with the worst possible value (ie. ∞ for minimization problems like VRP), except of course the attributes of the starting solution, which are initialized with the value of the objective function. In VRP this would be all the edges between adjecent customers in the routing plan. The algorithm is outlined in Algorithm 14.

To avoid looking through all the attributes, to see if a move improves on one of them, a short list of the worst attributes of the current solution is maintained by the `identifyWorstAttributes` function. When checking whether to accept a new move, the `accept` function compares the new objective function value with the list of worst attributes, and if it improves on at least one of the attributes, which is still in the solution after the neighbourhood move, the move is accepted. In the loop (line 6-10), the entire neighbourhood $\mathcal{N}$ is examined for acceptable moves, keeping track of the best one found. After the neighbourhood is exhaustively searched, the best move is made (line 12), and the attributes comprising this new solution are updated according to the objective value via the `updateAttributes` function.

The ABHC overcomes the problems of escaping local minima by allowing worsening solutions as long is at least one of its attributes improves. This allows it to explore the search landscape very extensively, while still favoring better solutions, due to the Best Fit nature of the algorithm. Every time the algorithm moves in search space, at least one property (read attribute) of the algorithm will measurably improve. This requirement for a strict improvement in every iteration ensures that the algorithm will terminate at some point, and furthermore that the algorithm will not visit the same solution twice.

The ABHC algorithm was chosen because it was shown to be very good at finding solutions for VRP, parameter free and relatively simple to implement. Due to the attribute principle, the algorithm seems to be efficient at escaping from local minima, which is an advantage when having clustered instances, in which there are farther between, and deeper local minima, than in a search landscape of a non-clustered instance.

Unlike other well known algorithms, like Simulated Annealing, Tabu Search, etc., where tuning is required to find the best setting for a specific type of problem or instance, the ABHC is parameter free. Thus as soon as one has defined the attributes that should be used for the given problem type, ABHC is applicable without tuning. While this is a very nice property in most cases, it also means that one cannot tune the speed of the algorithm. Where a low initial temperature would make the Simulated Annealing algorithm finish quicker, the basic ABHC has no parameters that can be set in a way to have the algorithm end faster.

Although no tuning is required, a few design choices still have to be made when implementing the algorithm. First of all, it has to be decided what property of the problem to use as attributes. Whittley and Smith [2004] used the arcs or edges between customers as attributes when applying ABHC to TSP. In Derigs and Kaiser [2007], experimentation with other properties as attributes for the VRP was done, but it was concluded that using arcs or edges yielded the best results. In terms of neighbourhood, relocate was chosen, and this is also supported by Derigs and Kaiser [2007] that showed the neighbourhood to be efficient for ABHC.

Comparison of the Offline ABHC to Solomon Benchmarks

To be able to examine the efficiency of the ABHC on VRPTW one can compare it to results found by other algorithms on known instances. For this, the Solomon instances were used (see

Section 5.1), as these are widely used to test VRPTW algorithms and are varied in terms of time windows and customer positioning.

For the comparison, the `Impact` algorithm was used for constructing the routes. No further improvement of the solutions done, other than the ABHC described in Algorithm 14.

As described in section 5.1 optimal solutions exist for many of the instances. Very tight upper and lower bounds have been found for many of the remaining, and heuristic results have been obtained for all the instances. In terms of optimal solutions and bounds the most recent solutions that could be found were reported in April 2008 by Jepsen et al. [2008].

In the context of this thesis, algorithms finding optimal solutions are considered infeasible, due to the time required for finding solutions. To be able to compare the implemented algorithm with alternatives the results of ABHC are also compared to the best solutions found by heuristics. The objective in the reported results is minimization of route length. Despite a considerable effort to find more recent reported results, the most recent results identified by heuristics that could be found are from the website of Solomon [2005], last updated March 2005.

In Table 6.3, the results of the ABHC are reported along with the optimal and heuristic solutions. For the instances where only bounds, rather than optimal solutions, were found, these are reported. In the two rightmost columns of the table, the percent-wise difference in solution quality of ABHC compared to the optimal and heuristic solution are reported. If no optimal solutions were available, but bounds were reported, the difference is calculated to the mean of the bounds.

As can be seen in the table, the algorithm is definitely competitive with the current heuristic methods. In most cases, ABHC improves on the solutions previously reported, and in 9 cases, it improves on the best solution obtained by considerable amounts (up to 13.71%).

Compared to the optimal solutions, the ABHC is within 5% of the optimal value. The running times for the optimal algorithms are normalized to a P4 3.0 GHz machine. The running time for ABHC is reported for a Intel Core2 6300 (1.86GHz) machine. As can be seen, in many cases for instances of type 1 (see section 5.1, page 33), the running times of optimal solutions are actually shorter than those of ABHC. Whether this is due to machine power or algorithm speed has not been possible to assess.

6.3.2 ABHC for the Consensus Algorithm and Speed-up Considerations

In terms of using the ABHC for the purpose of this thesis, a few further considerations are necessary. Unlike Derigs and Kaiser [2007], the objective in the version of VRP considered here, is to maximize the number of customers served, having a fixed number of vehicles. As a secondary objective is the minimization of the route length. This means that a valid solution could also contain a number of unassigned customers, and relocation of customers can therefore be between a route and the list of unassigned customer. This increases the size of the neighbourhood by some amount, but more importantly means besides having arcs as attributes, an attribute should also be associated with a customer being unassigned.

When using ABHC as a sub-procedure in the Consensus Algorithm, the algorithm will be run on routing plans that are partly fixated, due to parts of the customers being already visited. The effect of this on ABHC is simply that the size of neighbourhood for a given solution will be decreased, due to less customers being movable. In other words, the greater parts of the routing

Instance	**ABHC**		**Optimal**		**Heuristics**	**Opt. dev.**	**Heur. dev.**
	Distance	Time (s)	Distance	Time (s)	Distance	**(%)**	**(%)**
R101	1685.34	39.21	[1634.0; 1637.7]	1.87	1645.79	3.03	2.40
R102	1484.81	135.35	1466.6	4.39	1486.12	1.24	-0.09
R103	1233.93	238.19	1208.7	23.85	1292.68	2.09	-4.54
R104	1008.15	244.32	[971.3; 971.5]	23343.92	1007.24	3.78	0.09
R105	1405.34	70.59	[1355.2; 1355.3]	43.12	1377.11	3.70	2.05
R106	1267.67	187.38	1234.6	75.42	1251.98	2.68	1.25
R107	1081.12	172.27	[1064.3; 1064.6]	1310.3	1104.66	1.57	-2.13
R108	958.12	189.87	932.1	5911.74	960.88	2.79	-0.29
R109	1162.26	123.72	[1144.1; 1146.9]	1432.41	1194.73	1.46	-2.72
R110	1083.68	167.59	1068.0	1068.31	1118.59	1.47	-3.12
R111	1081.14	207.67	[1045.9; 1048.7]	83931.48	1096.72	3.23	-1.42
R112	974.83	184.37	[946.7; 948.6]	202803.94	982.14	2.87	-0.74
C101	828.94	9.18	827.3	3.02	828.94	0.20	0.00
C102	828.94	114.09	827.3	12.91	828.94	0.20	0.00
C103	828.07	335.32	826.3	33.89	828.06	0.21	0.00
C104	848.53	535.86	822.9	4113.09	824.78	3.11	2.88
C105	828.94	14.01	827.3	5.34	828.94	0.20	0.00
C106	828.94	72.55	827.3	7.15	828.94	0.20	0.00
C107	828.94	94.14	827.3	6.55	828.94	0.20	0.00
C108	828.94	195.91	827.3	14.46	828.94	0.20	0.00
C109	828.94	335.16	827.3	20.53	828.94	0.20	0.00
RC101	1699.77	42.03	1619.8	12.39	1696.94	4.94	0.17
RC102	1503.03	85.64	1457.4	76.69	1554.75	3.13	-3.33
RC103	1288.09	112.51	[1257.7; 1258.0]	2705.78	1261.67	2.40	2.09
RC104	1165.77	195.08	[1129.9; 1132.3]	65806.79	1135.48	3.07	2.67
RC105	1550.88	56.54	1513.7	26.73	1629.44	2.46	-4.82
RC106	1400.72	89.01	[1367.3; 1372.7]	15891.55	1424.73	2.24	-1.69
RC107	1261.56	105.53	1207.8	153.8	1230.48	4.45	2.53
RC108	1160.91	141.24	1114.2	3365.0	1139.82	4.19	1.85
R201	1187.28	224.7	1143.2	139.03	1252.37	3.86	-5.20
R202	1059.45	551.07	[1027.3; 1029.6]	8282.38	1191.7	3.01	-11.10
R203	891.67	816.59	870.8	54187.4	939.54	2.40	-5.10
R204	756.37	1413.86	-	-	825.52	-	**-8.38**
R205	969.75	425.53	-	-	994.42	-	**-2.48**
R206	914.11	719.14	-	-	906.14	-	0.88
R207	837.43	845.61	-	-	893.33	-	**-6.26**
R208	720.97	1317.33	-	-	726.75	-	**-0.80**
R209	883.53	563.65	854.8	78560.47	909.16	3.36	-2.82
R210	923.01	694.38	-	-	939.34	-	**-1.74**
R211	770.36	691.51	-	-	892.71	-	**-13.71**
C201	591.56	79.22	589.1	203.34	591.56	0.42	0.00
C202	591.56	353.1	589.1	3483.15	591.56	0.42	0.00
C203	591.17	620.55	588.7	13070.71	591.17	0.42	0.00
C204	590.6	1077.7	-	-	590.6	-	0.00
C205	588.88	227.04	586.4	416.56	588.88	0.42	0.00
C206	588.49	335.15	586.0	594.92	588.49	0.42	0.00
C207	588.29	344.23	585.8	1240.97	588.29	0.43	0.00
C208	588.32	453.97	585.8	555.27	588.32	0.43	0.00
RC201	1287.89	260.9	[1261.7; 1261.8]	229.27	1406.91	2.07	-8.46
RC202	1109.0	499.1	1092.3	312.57	1367.09	1.53	-18.88
RC203	953.24	655.93	923.7	14917.36	1049.62	3.20	-9.18
RC204	792.84	940.62	-	-	798.41	-	**-0.70**
RC205	1189.75	388.81	1154.0	221.24	1297.19	3.10	-8.28
RC206	1101.64	408.05	1051.1	339.69	1146.32	4.81	-3.90
RC207	972.21	586.16	-	-	1061.14	-	**-8.38**
RC208	787.33	760.68	-	-	828.14	-	**-4.93**

Table 6.3: Comparison of the ABHC, Optimal and Heuristic Solutions Found on the Solomon Benchmarks: The time unit is CPU-time in seconds. The results for optimal solutions and bounds are taken from Jepsen et al. [2008], published April 2008. The heuristic solution values were taken from Solomon [2005], last updated march 2005. The data was copied from the website April 20th, 2008. The rightmost columns specify percent-wise difference to optimal and heuristic solutions. If only bounds were available, rather than optimal solutions, the difference was taken to the mean of the bounds. In the results reported, the objective is minimization of distance.

plan that is fixated (ie. later on the day), the faster the ABHC generally is, due to the limited neighbourhood.

For the Online Stochastic Algorithms presented in section 4, the offline algorithm is only allowed to run for a limited amount of time, and therefore a mechanism for stopping the algorithm is necessary. The implementation of this is very simple: when the algorithm is out of time, it returns the current best solution (γ_r) found.

A potential problem with the ABHC, in terms of the Online Stochastic Algorithms, is that it might be too slow. During the run of the Consensus algorithm, a considerable amount of instance will have to be solved, and therefore a reasonably fast algorithm will have to be used. Although no parameters can be tuned for the basic ABHC to speed it up, there are a few changes that can be made to the algorithm, to attempt to make it faster. These changes will be described in the next three sections.

6.3.3 Improving Initial Solution Quality

The quality of the initial solution might have an effect on the efficiency of the algorithm. Derigs and Kaiser [2007] examine this, and concludes that ABHC seems robust in terms of solution quality regardless of the quality of the initial routing plan, but starting from a bad solution takes significantly more CPU-time/iteration than when a good construction heuristic is used.

As described above, the Impact algorithm followed by an Ejection Chain algorithm yielded very good results. But another option for attempting to create even better starting solutions, is to follow this by a run of a Best Fit local search. This was implemented using the relocation neighbourhood. The results are reported in Table 6.4. There is no general pattern, except that if one setting finds better results than the other, the time taken to do this is usually increased also. Although Derigs and Kaiser [2007] concludes that starting from a better solution saves significant amounts of CPU-time, this is when comparing a randomly generated solution to a run of a construction heuristic. The solution found by the Impact heuristic followed by an Ejection Chain is very good and so it does not make a significant difference in running time. An explanation for the variations in solution quality could be that the ABHC start from different solutions, and therefore does not necessarily visit the same parts of the search space. The decision was taken not to use a local search before running the ABHC algorithm.

6.3.4 ABHC with First Fit

Another option, also implemented by Nikolajsen [2009], is to change the algorithm to run by a First Fit principle. Rather than searching the entire neighbourhood in each iteration and taking the best acceptable improvement, one could do a move in the neighbourhood as soon as a move improving the best solution is found. This does not guarantee that the algorithm becomes faster, since using first fit makes it move slower towards local minima. On the other hand, in general the algorithm will search a smaller part of the neighbourhood, which is a substantial increase in speed for each iteration. In summary, using first fit means an increase in speed for each iteration, but at the cost of a less steep movement towards local minima. The algorithm is outlined in Algorithm 15. It is essentially the same as the regular ABHC except that it restarts its search every time a solution is found that improves on the overall best.

	Construction					Constr. and LS				
Instance	constr.		ABHC		cpu(s)	constr.		ABHC		cpu(s)
	u	length	u	length	(s)	u	length	u	length	(s)
R101	1	1959.6	0	1661.2	12.6	0	1753.4	0	1661.2	12.0
R102	1	1946.3	0	1507.0	56.7	0	1588.8	0	1488.3	78.3
R103	0	1797.6	0	1250.6	122.3	0	1368.4	0	1249.3	126.3
R104	4	1456.8	0	993.3	97.3	2	1146.1	0	993.3	96.2
R105	0	1734.8	0	1415.4	19.8	0	1533.3	0	1405.1	18.9
R106	1	1800.6	0	1320.5	57.4	1	1461.0	0	1279.0	70.1
R107	4	1441.3	0	1095.0	56.6	3	1252.0	0	1095.0	56.3
R108	8	1327.8	0	949.5	83.3	2	1057.0	0	949.5	82.3
R109	2	1670.2	0	1165.0	58.2	2	1460.2	0	1165.0	58.0
R110	5	1533.4	0	1076.2	78.8	3	1393.6	0	1076.2	78.6
R111	5	1445.1	0	1131.0	93.4	5	1270.2	0	1131.0	92.8
R112	1	1402.7	0	982.0	98.2	1	1198.5	0	982.0	97.8
R201	0	2490.9	0	1235.1	116.7	0	1382.8	0	1192.9	108.6
R202	0	2147.7	0	1110.8	290.5	0	1383.0	0	1110.8	290.3
R203	0	2105.3	0	909.1	534.1	0	1269.1	0	909.1	537.2
R204	0	1864.7	0	755.6	895.0	0	968.8	0	755.6	891.2
R205	0	2295.7	0	996.8	250.7	0	1171.1	0	996.8	273.5
R206	0	2334.6	0	894.3	628.2	0	1159.5	0	894.3	585.0
R207	0	1937.7	0	815.0	779.8	0	960.6	0	812.8	784.2
R208	0	1912.1	0	707.7	898.5	0	885.5	0	707.7	895.2
R209	0	2506.8	0	864.1	353.8	0	1041.9	0	864.1	350.4
R210	0	2667.3	0	932.8	415.5	0	1150.6	0	932.8	412.2
R211	0	2385.6	0	763.9	592.7	0	933.5	0	763.9	593.0
C101	6	1303.5	5	1115.3	3.3	5	1221.8	5	1115.3	2.5
C102	1	1990.6	0	995.0	28.5	1	1639.9	0	995.0	28.5
C103	0	2299.3	0	859.4	148.6	0	1496.1	0	859.4	147.2
C104	0	1978.4	0	824.8	363.5	0	1337.6	0	824.8	362.9
C105	7	1701.3	3	1253.8	10.9	7	1640.6	5	1341.9	9.2
C106	5	1903.7	0	893.1	15.2	5	1798.4	0	893.1	15.4
C107	3	1902.0	0	969.0	18.2	3	1679.4	0	969.0	18.2
C108	3	1714.5	0	1051.6	28.1	3	1561.1	0	1051.6	28.2
C109	0	1936.4	0	828.9	131.6	0	1813.1	0	828.9	132.1
C201	0	1297.0	0	591.6	44.2	0	860.6	0	629.7	30.6
C202	4	1381.1	0	985.6	14.8	4	1336.5	0	985.6	14.8
C203	0	2259.7	0	620.3	372.5	0	1398.4	0	620.3	371.5
C204	6	1557.4	0	590.6	520.7	2	1133.4	0	590.6	513.7
C205	1	1013.8	1	770.9	12.5	1	964.9	1	766.6	12.8
C206	0	945.4	0	672.1	21.9	0	903.9	0	672.1	22.1
C207	1	1164.5	0	588.3	52.4	1	1038.6	0	588.3	52.4
C208	2	1169.8	0	678.5	46.5	0	1002.9	0	678.5	42.8
RC101	3	2056.8	0	1700.3	13.3	3	1855.8	0	1700.3	12.3
RC102	1	2055.4	0	1478.0	34.2	0	1911.7	0	1478.0	36.0
RC103	3	1727.1	0	1321.1	63.7	2	1345.5	0	1321.1	66.2
RC104	10	1430.2	0	1158.5	92.3	7	1332.2	0	1158.5	91.7
RC105	3	1999.0	0	1644.3	10.6	2	1843.7	0	1644.3	10.6
RC106	2	1732.9	0	1430.6	26.4	0	1499.6	0	1437.7	19.8
RC107	5	1651.5	0	1321.0	35.0	3	1408.2	0	1307.3	50.3
RC108	4	1566.8	0	1151.9	49.8	3	1445.2	0	1151.9	49.6
RC201	0	2498.0	0	1369.8	94.8	0	1687.5	0	1369.8	98.5
RC202	0	2478.1	0	1234.2	234.2	0	1511.4	0	1241.6	223.1
RC203	0	2554.4	0	1003.8	393.2	0	1219.9	0	1003.8	394.7
RC204	0	2345.5	0	804.2	716.9	0	1063.7	0	804.2	712.3
RC205	0	2750.9	0	1260.6	220.1	0	1504.8	0	1260.6	222.4
RC206	0	2608.5	0	1125.2	180.9	0	1420.3	0	1111.0	174.0
RC207	0	3077.3	0	970.8	494.7	0	1301.3	0	970.8	481.1
RC208	0	2685.7	0	793.2	456.8	0	1038.3	0	793.2	455.1

Table 6.4: Comparison of the ABHC Run With Different Quality of Starting Solutions: *ABHC normal* is run with the Impact construction algorithm only. *ABHC improved* has the Impact solution improved by a Best Fit local search with the relocate neighbourhood. The time unit is CPU-time in seconds. For ABHC columns, the time given is the total running time for construction heuristic *and* ABHC. The columns denoted *u* denote the number of unserved customers.

Algorithm 15: ABHC algorithm for VRP with First Fit

```
Data: Routing Plan $\gamma$
Result: Routing Plan $\gamma_r$
$\gamma_r \leftarrow \gamma$
while true do
    IdentifyWorstAttributes($\gamma$)
    foreach $\gamma_t \in \mathcal{N}(\gamma)$ do
        if Accept($\gamma, \gamma_t$) then
            $\gamma \leftarrow \gamma_t$
            UpdateAttributes($\gamma$)
            if $\gamma < \gamma_r$ then
                $\gamma_r \leftarrow \gamma$
                restart while
            end
        end
    end
    break while
end
```

Results of First Fit

The ABHC using the First Fit principle was compared with ABHC using Best Fit. The results are reported in Table 6.5. The three rightmost columns show the differences, where bold results denote those in which Best Fit gave the best results. Although the solutions are not unambiguous, the choice of algorithm setting to use is Best Fit. Amongst other, this is based on Best Fit finding two solutions with less unserviced customers, as opposed to First Fit, not finding any solutions with fewer unserved customers than Best Fit.

6.3.5 Other Ways of Speeding Up ABHC

In the basic algorithm, all the attributes are initialized with ∞ except, of course, those that are part of the starting solution. This allows the algorithm to traverse even the global maximum, and thereby also reach different local minima. While this is desirable for searching as much of the search landscape as possible, it also potentially increases the running time of the algorithm, due to the time consumed by searching irrelevant parts of the search landscape. Instead, one could initialize the attributes with, for example, the objective value of the initial solution x. This would mean that the algorithm would not be allowed to traverse any solutions with an objective value greater than x. This obviously limits the ways the algorithm can traverse the search landscape, and therefore could also limit the running time of the algorithm. The pitfall of this change, is that having an x of too low value, would limit the searched landscape by too much, and having a very large x would have too little effect to affect the execution time.

For testing this, runs of the ABHC was made with 4 different starting values. Calling the

	Best Fit			**First Fit**			**Diff.**		
Instance	u	length	cpu(s)	u	length	cpu(s)	u	length	cpu(s)
R101	0	1661.21	13.04	0	1676.21	12.70	0	**-15.00**	0.34
R102	0	1507.03	60.41	0	1482.69	85.86	0	24.34	**-25.45**
R103	0	1250.59	127.32	0	1244.33	85.78	0	6.26	41.54
R104	0	993.30	100.81	0	990.82	111.80	0	2.48	**-10.99**
R105	0	1415.37	20.58	0	1478.80	10.73	0	**-63.43**	9.85
R106	0	1320.47	59.81	0	1262.38	103.27	0	58.09	**-43.46**
R107	0	1094.99	59.22	0	1104.62	81.97	0	**-9.63**	**-22.75**
R108	0	949.54	86.44	0	954.38	66.39	0	**-4.83**	20.05
R109	0	1164.99	60.55	0	1183.66	62.62	0	**-18.67**	**-2.07**
R110	0	1076.16	81.80	0	1090.69	71.24	0	**-14.53**	10.56
R111	0	1131.04	96.88	0	1093.80	102.17	0	37.25	**-5.29**
R112	0	981.98	101.13	0	975.14	108.17	0	6.84	**-7.04**
R201	0	1235.11	117.30	0	1221.88	92.29	0	13.23	25.01
R202	0	1110.81	291.99	0	1110.17	254.01	0	0.64	37.98
R203	0	909.09	531.33	0	901.20	655.14	0	7.89	**-123.81**
R204	0	755.61	918.19	0	763.38	768.92	0	**-7.78**	149.27
R205	0	996.76	257.33	0	979.60	282.51	0	17.16	**-25.18**
R206	0	894.33	643.46	0	909.12	506.77	0	**-14.79**	136.69
R207	0	814.99	785.11	0	824.06	687.14	0	**-9.07**	97.97
R208	0	707.71	905.11	0	730.32	979.72	0	**-22.61**	**-74.61**
R209	0	864.14	354.97	0	868.58	375.37	0	**-4.44**	**-20.40**
R210	0	932.83	414.83	0	930.40	467.66	0	2.43	**-52.83**
R211	0	763.93	588.74	0	757.86	668.92	0	6.07	**-80.18**
C101	5	1115.31	3.30	5	1115.31	2.44	0	0.00	0.86
C102	0	995.01	29.51	0	1141.87	46.99	0	**-146.86**	**-17.48**
C103	0	859.37	147.84	0	828.07	121.13	0	31.30	26.71
C104	0	824.78	359.72	0	852.59	299.85	0	**-27.81**	59.87
C105	3	1253.75	10.73	5	1401.45	7.38	**-2**	-	3.35
C106	0	893.14	15.12	0	1181.40	15.82	0	**-288.27**	**-0.70**
C107	0	968.96	18.58	0	1067.77	12.23	0	**-98.81**	6.35
C108	0	1051.56	28.66	0	959.74	34.03	0	91.82	**-5.37**
C109	0	828.94	130.05	0	881.05	114.17	0	**-52.11**	15.88
C201	0	591.56	44.36	0	591.56	10.34	0	0.00	34.02
C202	0	985.63	14.81	0	964.18	19.13	0	21.45	**-4.32**
C203	0	620.30	377.55	0	591.17	434.98	0	29.12	**-57.43**
C204	0	590.60	525.00	0	823.15	298.67	0	**-232.55**	226.33
C205	1	770.90	12.59	1	770.90	12.46	0	0.00	0.13
C206	0	672.10	22.03	0	588.49	37.13	0	83.60	**-15.10**
C207	0	588.29	52.66	0	588.29	43.11	0	0.00	9.55
C208	0	678.50	46.39	0	678.50	53.22	0	0.00	**-6.83**
RC101	0	1700.30	13.14	0	1715.34	12.04	0	**-15.04**	1.10
RC102	0	1477.96	33.71	0	1494.05	69.83	0	**-16.09**	**-36.12**
RC103	0	1321.09	62.80	0	1278.30	51.39	0	42.79	11.41
RC104	0	1158.52	90.97	0	1183.88	69.34	0	**-25.37**	21.63
RC105	0	1644.31	10.32	1	1716.16	10.57	**-1**	-	**-0.25**
RC106	0	1430.55	25.85	0	1406.81	16.70	0	23.74	9.15
RC107	0	1320.98	33.92	0	1294.32	31.41	0	26.66	2.51
RC108	0	1151.89	48.75	0	1163.76	57.24	0	**-11.88**	-8.49
RC201	0	1369.85	94.05	0	1405.97	76.52	0	-36.12	17.53
RC202	0	1234.16	233.35	0	1197.91	262.32	0	36.25	**-28.97**
RC203	0	1003.80	390.61	0	969.97	553.39	0	33.83	**-162.78**
RC204	0	804.18	730.25	0	791.40	740.60	0	12.78	**-10.35**
RC205	0	1260.64	223.45	0	1265.62	186.23	0	**-4.98**	37.22
RC206	0	1125.23	184.04	0	1126.25	193.73	0	**-1.02**	**-9.69**
RC207	0	970.78	494.97	0	996.38	433.18	0	**-25.60**	61.79
RC208	0	793.19	456.94	0	782.66	507.96	0	10.53	**-51.02**

Table 6.5: Comparison of the ABHC using First Fit and Best Fit: The label *u* in the column header, denotes unserved customers. The rightmost column reports the difference in the quality of the two settings. Results reported in bold are those in which Best Fit performed best.

objective value of the starting solution a, tests was made with initializing the attributes to $0.95 \cdot a$, $1.00 \cdot a$, $1.05 \cdot a$ and $1.15 \cdot a$. These were compared to runs with the regular initial value of ∞. The starting solution was found by a run of `Impact` followed by the Ejection Chain algorithm. The result can be found in Table 6.6.

As can be seen, a setting of $0.95 \cdot a$ objective value is extremely fast, but the results are equally bad. Oddly, for the rest of the settings the running times are worse than those of the ∞ setting, even though this finds betters solutions. Further tests were made with settings of $1.50 \cdot a$ and $2.00 \cdot a$, and although these yielded better results than the initial solutions displayed in the table, the solutions of ∞ were still superior in terms of both time and quality. It has not been possible to diagnose why this happens. It might have something to do with the freedom of movement in the search landscape. The standard ABHC is allowed to move freely in the search landscape, thus very quickly locating good local optimums. This freedom is greatly limited by having an lower attribute value, since ABHC have to find alternative ways from minima to minima, not exceeding the attribute value. This means that the movement in the search landscape will be slower, and attributes not in the start solution will be involved in a solution very late in the search process. Alternatively, with the standard ABHC, the attributes might be involved in a solution very fast, due to the freedom of movement.

Obviously, the choice for initial value of edges is ∞.

6.4 Iterated Local Search (ILS)

Although ABHC finds good solutions, its long runtime potentially makes it unsuitable as a sub-procedure for the online stochastic algorithms described above - solving the dynamic VRPTW would simply take too long with full runs of ABHC. Rather, an efficient algorithm is necessary, since the runtime has to be short for it to be usable.

The choice of `Iterated Local Search` (ILS) [Hoos and Stützle, 2005] is amongst other based on this. ILS is a meta heuristic based on regular local search. The idea is to solve an instance with a local search to reach a local optimum. As an escape strategy, the solution is permuted and a new local search is made. This is continued until some criteria (eg. a time limit) is met. At this point, the algorithm returns the best solution found. The permutation allows the algorithm to escape the local minimum so the following local search is able to reach a new local minimum.

Although briefly examined by Van Hentenryck and Bent [2006], it is not clear what the best balance is between speed in offline algorithm and quality of offline solutions, when using it for the Online Stochastic Algorithms. The speed affects the number of instances that can be sampled and solved and the number of optimizations on each instance. The fact that a run of iterated local search can be made arbitrarily long (by allowing arbitrarily many search/permutation cycles), makes ILS very suitable for examining the above mentioned balance. The fastest possible run of ILS is to simply do one local search and return the solution as the result. This is effectively the same amount of time as a local search. For each permutation/local search cycle, the algorithm will take longer, but also have an increased chance of returning an improved result.

Instance	Sol. Value·0.95			Sol. Value·1			Sol. Value·1.05			Sol. Value·1.15			∞		
	u	length	cpu(s)	u	length	cpu(s)	u	length	cpu(s)	u	length	cpu(s)			
C101	5	1221.79	2.10	5	1142.97	3.84	5	1142.97	5.00	5	1142.97	6.62	5	1115.31	2.34
C102	1	1990.56	0.02	1	1426.83	10.72	1	1426.83	36.08	1	1426.83	38.26	0	995.01	28.74
C103	0	2299.31	0.02	0	870.32	205.14	0	870.32	442.87	0	870.32	459.93	0	859.37	149.03
C104	0	1978.38	0.03	0	824.78	481.36	0	824.78	1237.42	0	824.78	1319.75	0	824.78	365.99
C105	7	1701.27	0.03	7	1638.30	2.51	7	1638.30	5.21	7	1638.30	5.40	3	1253.75	10.75
C106	5	1903.67	0.02	5	1798.38	1.78	5	1798.38	2.50	5	1798.38	2.62	0	893.14	15.30
C107	3	1901.96	0.02	3	1633.59	3.78	3	1633.59	5.75	3	1633.59	7.14	0	968.96	18.25
C108	3	1714.50	0.03	0	1051.56	128.42	0	1051.56	174.85	0	1051.56	188.30	0	1051.56	28.61
C109	0	1936.40	0.03	0	828.94	166.81	0	828.94	496.71	0	828.94	531.12	0	828.94	132.62
C201	0	715.07	20.82	0	715.07	7.96	0	715.07	19.25	0	715.07	20.21	0	591.56	43.12
C202	4	1381.11	0.02	4	1336.03	1.71	0	985.63	77.52	0	985.63	81.58	0	985.63	14.54
C203	0	2259.68	0.02	0	620.30	447.25	0	620.30	750.19	0	620.30	994.33	0	620.30	373.41
C204	6	1557.39	0.01	0	590.60	1162.65	0	590.60	1552.41	0	590.60	1832.28	0	590.60	527.03
C205	1	1013.79	0.02	1	964.86	1.27	1	964.86	3.27	1	964.86	3.53	1	770.90	12.56
C206	0	945.44	0.02	0	672.10	21.35	0	672.10	44.73	0	672.10	47.56	0	672.10	21.89
C207	1	1164.46	0.02	0	588.29	73.69	0	588.29	175.02	0	588.29	184.84	0	588.29	52.13
C208	2	1169.78	0.03	0	678.50	82.82	0	678.50	237.29	0	678.50	245.81	0	678.50	46.41
R101	1	1959.64	0.18	0	1661.21	21.70	0	1661.21	53.38	0	1661.21	57.89	0	1661.21	12.15
R102	1	1946.28	0.02	0	1507.03	109.71	0	1507.03	111.36	0	1507.03	111.32	0	1507.03	60.68
R103	0	1797.63	0.03	0	1279.13	113.80	0	1279.13	252.82	0	1279.13	391.24	0	1250.59	130.28
R104	4	1456.85	0.02	0	993.30	355.35	0	993.30	614.41	0	993.30	604.44	0	993.30	103.28
R105	0	1734.82	0.02	0	1415.37	29.52	0	1415.37	68.03	0	1415.37	70.82	0	1415.37	20.93
R106	1	1800.63	0.02	0	1320.47	86.13	0	1320.47	208.56	0	1320.47	221.10	0	1320.47	60.74
R107	4	1441.31	0.02	0	1094.99	236.53	0	1094.99	318.10	0	1094.99	328.00	0	1094.99	60.37
R108	8	1327.77	0.02	0	949.54	317.32	0	949.54	445.59	0	949.54	445.71	0	949.54	88.10
R109	2	1670.24	0.04	0	1164.99	119.50	0	1164.99	376.05	0	1164.99	395.40	0	1164.99	61.31
R110	5	1533.38	0.02	0	1076.16	309.42	0	1076.16	424.45	0	1076.16	459.40	0	1076.16	83.78
R111	5	1445.10	0.02	0	1131.04	345.12	0	1131.04	511.06	0	1131.04	606.87	0	1131.04	97.80
R112	1	1402.74	0.02	0	981.98	181.81	0	981.98	412.83	0	981.98	440.84	0	981.98	103.48
R201	0	2490.93	0.14	0	1246.64	108.59	0	1246.64	246.04	0	1246.64	357.32	0	1235.11	115.42
R202	0	2147.67	0.04	0	1110.81	412.77	0	1110.81	812.95	0	1110.81	1228.36	0	1110.81	289.00
R203	0	2105.30	0.03	0	909.09	699.64	0	909.09	1174.84	0	909.09	1668.70	0	909.09	525.90
R204	0	1864.73	0.02	0	755.61	1080.07	0	755.61	2164.19	0	755.61	2243.73	0	755.61	909.69
R205	0	2295.75	0.02	0	996.76	340.87	0	996.76	652.22	0	996.76	643.75	0	996.76	256.09
R206	0	2334.61	0.02	0	894.33	778.49	0	894.33	1388.98	0	894.33	1886.00	0	894.33	635.13
R207	0	1937.67	0.01	0	814.99	897.43	0	814.99	1393.04	0	814.99	1444.39	0	814.99	789.06
R208	0	1912.13	0.04	0	707.71	1076.80	0	707.71	1959.21	0	707.71	2054.67	0	707.71	917.18
R209	0	2506.80	0.02	0	864.14	438.49	0	864.14	802.15	0	864.14	834.44	0	864.14	355.68
R210	0	2667.28	0.01	0	932.83	514.32	0	932.83	910.91	0	932.83	1255.44	0	932.83	417.55
R211	0	2385.57	0.02	0	763.93	678.58	0	763.93	1070.47	0	763.93	1102.87	0	763.93	593.57
RC101	3	2056.83	0.20	0	1700.30	43.43	0	1700.30	77.81	0	1700.30	62.57	0	1700.30	12.81
RC102	1	2055.36	0.02	0	1545.80	38.26	0	1545.80	83.00	0	1545.80	86.97	0	1477.96	36.19
RC103	3	1727.08	0.00	0	1321.09	195.97	0	1321.09	197.33	0	1321.09	375.62	0	1321.09	66.86
RC104	10	1430.22	0.03	0	1158.52	314.87	0	1158.52	521.19	0	1158.52	462.49	0	1158.52	96.57
RC105	3	1999.04	0.03	0	1644.31	23.73	0	1644.31	43.58	0	1644.31	37.70	0	1644.31	11.02
RC106	2	1732.95	0.03	0	1446.01	36.78	0	1446.01	98.30	0	1446.01	102.60	0	1430.55	27.73
RC107	5	1651.51	0.02	0	1320.98	123.47	0	1320.98	182.80	0	1320.98	226.10	0	1320.98	36.28
RC108	4	1566.82	0.03	0	1151.89	171.20	0	1151.89	308.32	0	1151.89	306.07	0	1151.89	52.38
RC201	0	2498.03	0.12	0	1369.85	144.92	0	1369.85	300.15	0	1369.85	299.99	0	1369.85	92.16
RC202	0	2478.15	0.02	0	1234.16	350.19	0	1234.16	673.19	0	1234.16	1018.23	0	1234.16	233.29
RC203	0	2554.39	0.02	0	1003.80	474.47	0	1003.80	802.27	0	1003.80	818.73	0	1003.80	391.68
RC204	0	2345.47	0.02	0	804.18	788.85	0	804.18	1158.29	0	804.18	1609.18	0	804.18	727.06
RC205	0	2750.94	0.02	0	1260.64	334.73	0	1260.64	695.13	0	1260.64	1004.92	0	1260.64	223.43
RC206	0	2608.50	0.01	0	1125.23	251.69	0	1125.23	494.43	0	1125.23	526.24	0	1125.23	183.57
RC207	0	3077.34	0.02	0	970.78	626.40	0	970.78	1091.04	0	970.78	1594.38	0	970.78	502.24
RC208	0	2685.74	0.02	0	793.19	592.65	0	793.19	996.53	0	793.19	1444.73	0	793.19	463.61

Table 6.6: Comparison of the ABHC with Different Initial Values for attributes: *Sol. Value* is the objective value of the starting solution. The ABHC was run with this value multiplied by 0.95, 1, 1.05 and 1.15, and also with the original setting of ∞. The starting solution was created using a run of `Impact` followed by a run of the Ejection Chain algorithm. Label *u* in the column header denotes unserved customers.

.

6.4.1 Base Algorithm

As described above, the basic idea of Iterated Local Search is to escape a local optimum, by permuting the solution followed by a new local search. Assuming the solution was permuted into a suitable region of the search space, there is a chance that the algorithm might find a new local optimum. The base algorithm is described in Algorithm 16 below, while the implementation of the actual permutation will be described afterwards.

Algorithm 16: IteratedLocalSearch

```
Data: routingPlan $\gamma$, timeLimit
Result: Routing plan $\gamma_b$

1 $\gamma_b \leftarrow \gamma$;
2 while time spent $<$ timeLimit do
3     $\gamma \leftarrow$ permute($\gamma_b$)
4     $\gamma \leftarrow$ localSearch($\gamma$)
5     if $w(\gamma) < w(\gamma_b)$ then
6         $\gamma_b \leftarrow \gamma$
7     end
8 end
9 return $\gamma_b$
```

The algorithm takes a starting solution and a time limit as arguments. This solution is saved as the current best (line 1). Following this, the main loop (line 2-8) is entered, running for as long as allowed by the `timeLimit` argument. The core of the algorithm is line 3 and 4, in which the current best solution is first permuted, and then guided to a local optimum by a local search. If an improvement is found, this new solution is saved as the best (line 5-7). If the time limit has not been reached, the while loop ensures the continued running of the algorithm. When the time limit is reached, the loop is terminated, and the best solution found thus far is returned (line 9).

This is the most basic version of the algorithm, and it could be extended in several ways. One common modification is to improve the condition of the while loop. This could be done by checking the number of iterations done without finding improvements. If the algorithm is not able to find improvements in a long series of permutation/search cycles, chances are it is not worth the time to keep trying, at least not using the same permutation method.

As can be seen above, the basic algorithm is very simple, and the actual work is done in its two important subroutines; the permutation and local search. The choice of local search will be described in section 6.4.2. The options for permuting the solution is described below.

Permutation by Neighbourhood If the local search algorithm, used in ILS, is using a single neighbourhood, the permutation could consist of a series of moves in another neighbourhood. These moves should not be restricted by a requirement of improving the objective function, but rather serve as a way of scrambling the solution in a way the local search neighbourhood is not able to. Since the defined neighbourhoods used in this project only move between feasible solutions, the permutation will result in a feasible solution, and hence the local search will of course also result in one.

A potential problem with this permutation, is that the permuting neighbourhood might not be able to scramble the solution sufficiently. As an example, we let the `relocation` neighbourhood be used in the local search. If the found solution is very tight in terms of the slack in visiting times, so each customer c_i is visited just before $l(c_i)$, this allows very few, if any, valid moves for the `exchange` neighbourhood. This means that the permutation will not be scrambled very much, and consequently ILS is less likely to explore many local optima. Furthermore, finding valid moves for permutation might be time consuming, which is not desirable in the permutation phase of the algorithm.

Permutation by Removal Another option is, that given a solution, some percentage of the customers are removed from the tours, and put into the pool of unassigned customers. This ensures that the remaining assigned customers are still valid in terms of their time windows. And in the local search following the randomization, the unassigned customers can be reassigned to vehicles.

Of course, one should ensure that a feasible solution is indeed reachable after this kind of permutation. If objective function (2.6) or (2.7), page 8, is used, a feasible solution is easily reachable if using the `relocate` neighbourhood in the local search. Since we have an unlimited number of vehicles available, one can simply take each customer from the set of unserved customers, and assign it to a new route. While this would result in a very bad solution, it is nevertheless feasible. Using the `exchange` neighbourhood, on the other hand, does not allow us to reach a feasible solution, so this is not usable in combination with permuting by removal of customers. If the objective (2.9), page 9, is used, the solution found by the permutation is feasible, since unserved customers are allowed according to this.

The point of the permutation in ILS is to scramble the solution sufficiently to reach a different part of the search space, where another local minimum can be found, but not as much as to be similar to a Random Restart heuristic, in which the local search start from a completely random solution. In terms of the permutation by removal; if too few customers are removed from the routing plan, the local search would be likely to insert the unrouted customers in the same places, reaching the same local optimum and making the randomization pointless. Removing too many customers could be too close to a total randomization, since there is too great a freedom in placement of customers, making the algorithm slow and inefficient, and effectively very similar to a random restart local search. The main point in the permuting part of ILS is to guide the algorithm to different positions in the search space, not to limit its exploration to a too small part of it. Therefore, some sort of tuning in the number of customers to be removed has to be done.

Permutation by Removal and Neighbourhood Yet another option would be to use a combination of the two permutations described above. First: remove of some of the customers in the routes, then follow this by a series of moves in the neighbourhood not used by the local search. The removal of customers allows a greater freedom for the neighbourhood to scramble the solution, reducing the problem with too tight solutions described above. Using a different neighbourhood for the permutation scrambles the solution further, while maintaining valid time windows. Hopefully, this forces the local search to examine a different part of the search space, than it would if the simple removal of customers was used. The time used for this permutation could be problematic, but having having more freedom from the removed customers, is likely to make valid moves in the permuting neighbourhood easier to find, thus decreasing the time spent by this part. Like permutation by removal, this type of permutation is only valid when using the relocate neighbourhood in the local search fase, because we want to be able to find a feasible solution when performing local search.

6.4.2 Tuning

For the local search sub-procedure, Best Fit is used. This simply searches the entire neighbourhood of the current routing plan, and moves to the best neighbour. Due to inability of exhange to add new unassigned customers to the routing plan, the Relocate neighbourhood is used by the Best Fit local search.

To find the best type of permutation, the three permutations described above were tuned individually. This was done on six instances, one from each class of Solomon instances, selected at random. As the ILS might be run in a range of different time spans, the tuning was done on a 30 CPU second time horizon, but results were reported as soon as they were found. This allows the examination of how fast good results are found for the different settings. The tuning will be described in the following.

Due to the tuning being inconclusive, a race was set up comparing all settings on all permutations, on a 10 CPU second time horizon. This will be described after the individual tuning of the permutation types.

Tuning Permutation by Neighbourhood

As described above, 6 instances were used to tune the permutation parameters. An unserved customer contribute with 1000 units to the objective function, equal to adding a route length of 1000. The number of moves made in each permutation were selected to be 5, 15, 25 and 35. For each move, the entire neighbourhood is searched for feasible moves, and a random of these is selected. Since the local search used the Relocate neighbourhood, the permutation was done using the exchange neighbourhood.

The results are shown in Figure 6.4. As can be seen, no setting was overall superior. All settings find good solutions on some instances, while being inferior on others. To find the best setting further testing is needed. This is done when racing the different permutation types below.

Tuning Permutation by Removal

The tuning of permutation by removal of customers was done in the same way as with permutation by neighbourhood, described above. An unserved customer contribute 1000 units to the objective function. Settings was tested with 5, 10, 15, 20 and 25% of the routed customers being removed. The results are compiled in Figure 6.5. Overall, the setting of 25% seems to be performing well, except on the C107 instance, on which the 15% setting is superior. Also, the setting of 5% seems to perform rather poorly in general. The explanation for this, is when removing only 5% of the customers, the local search simply finds the same local minimum. Or in other words, the solution is not sufficiently permuted to allow the local search to reach new local minima.

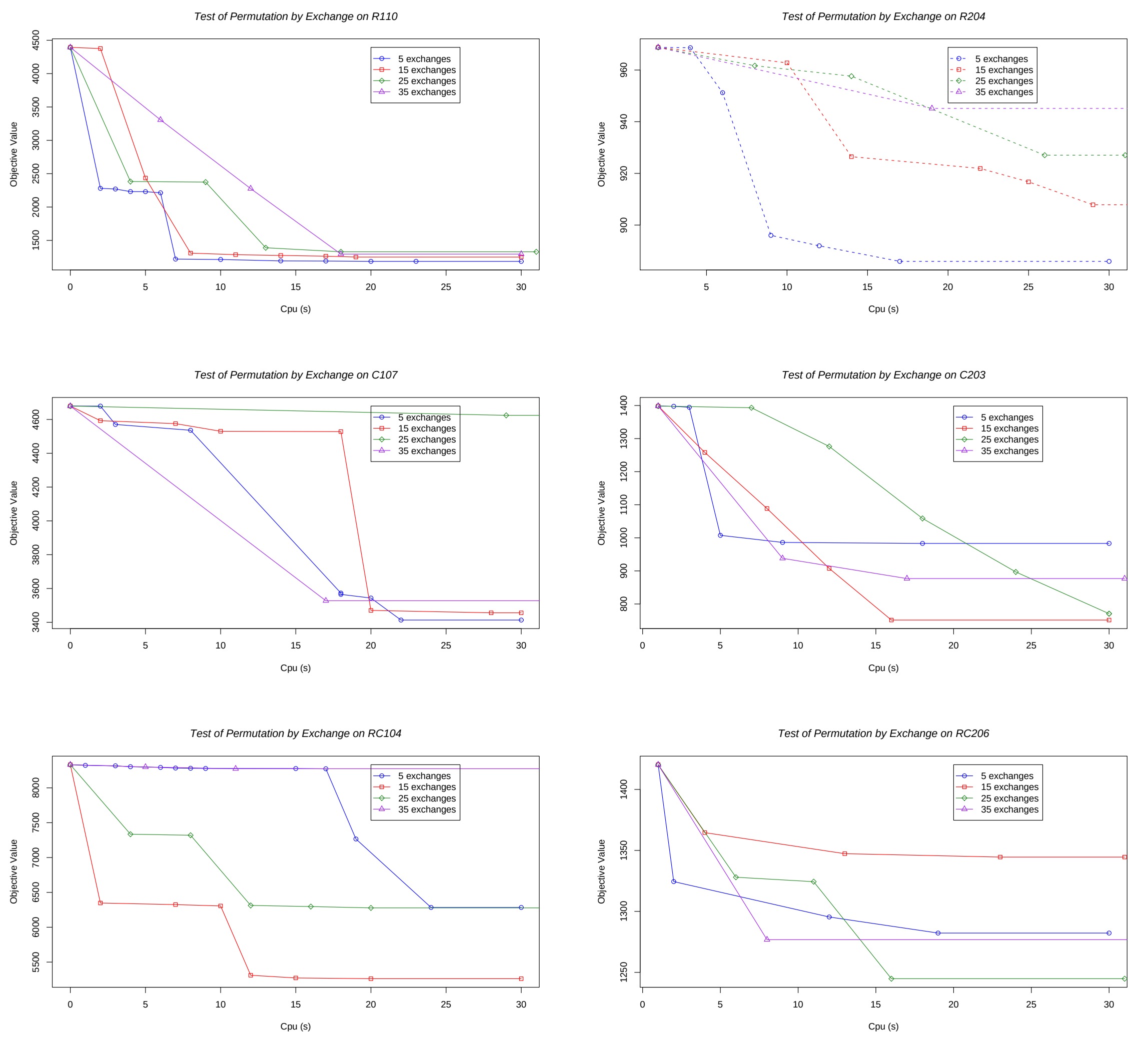


Figure 6.4: Permutation by Neighbourhood: Graphs of the ILS algorithm run on one Solomon instance of each class. For each instance, four settings for number of moves to do were tested: 5, 15, 25 and 35. Larger versions of the graphs can be found in Appendix A.1.1.

Tuning Permutation by Removal and Neighbourhood

As described above, the problem with permuting by removal, could be that the solutions are not sufficiently permuted, and the following local search simply reaches the same local minimum. Similarly, in a tight plan, there might not be room for very many neighbouring moves, resulting in too little permutation being done. The third option is the combination of these two permutations, in which a percent of the routed customers are removed, followed by b moves in the neigbhourhood. This was tested with the settings of $(a = 5, b = 5)$, $(a = 5, b = 15)$,

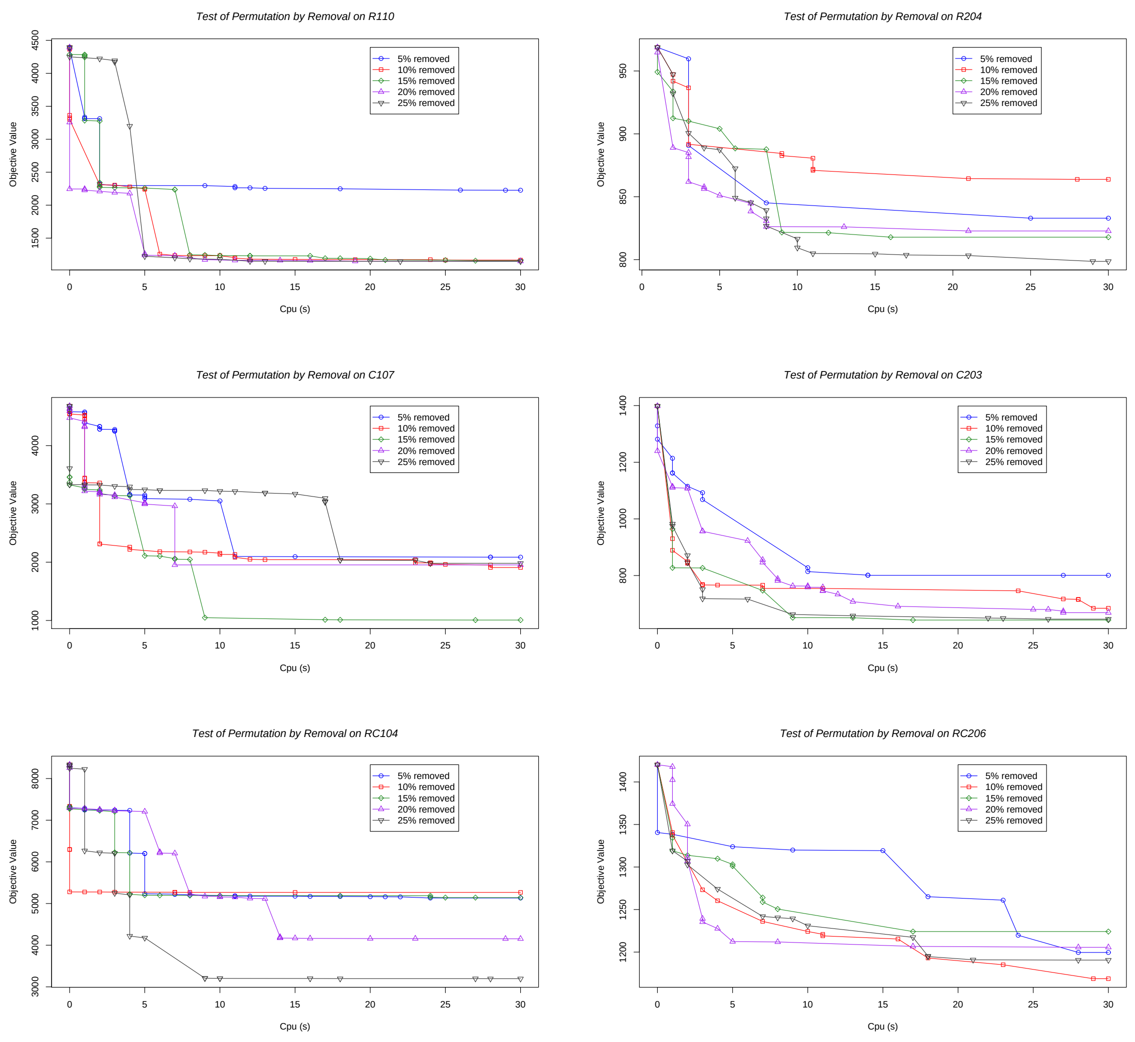


Figure 6.5: Permutation by Removal: Graphs of the ILS algorithm running on one Solomon instance of each class. For each instance, five settings for percentage of customers to remove were tested: 5, 10, 15, 20 and 25%. Larger versions of the graphs can be found in Appendix A.1.2

$(a = 10, b = 5)$ and $(a = 10, b = 15)$. The results are compiled in Figure 6.6.

Again, it is hard to point out a superior setting. Although a setting of $(a = 10, b = 15)$ seems to perform well in general it fails to produce good results on C107. Similarly, a setting of $(a = 5, b = 5)$ seems to perform well, except for the instances R204 and RC206. Since these are the two extremes of the parameter settings, it could indicate that the results are very dependent on instance type. In any case, further testing needs to be done on the parameters, and this is done below.

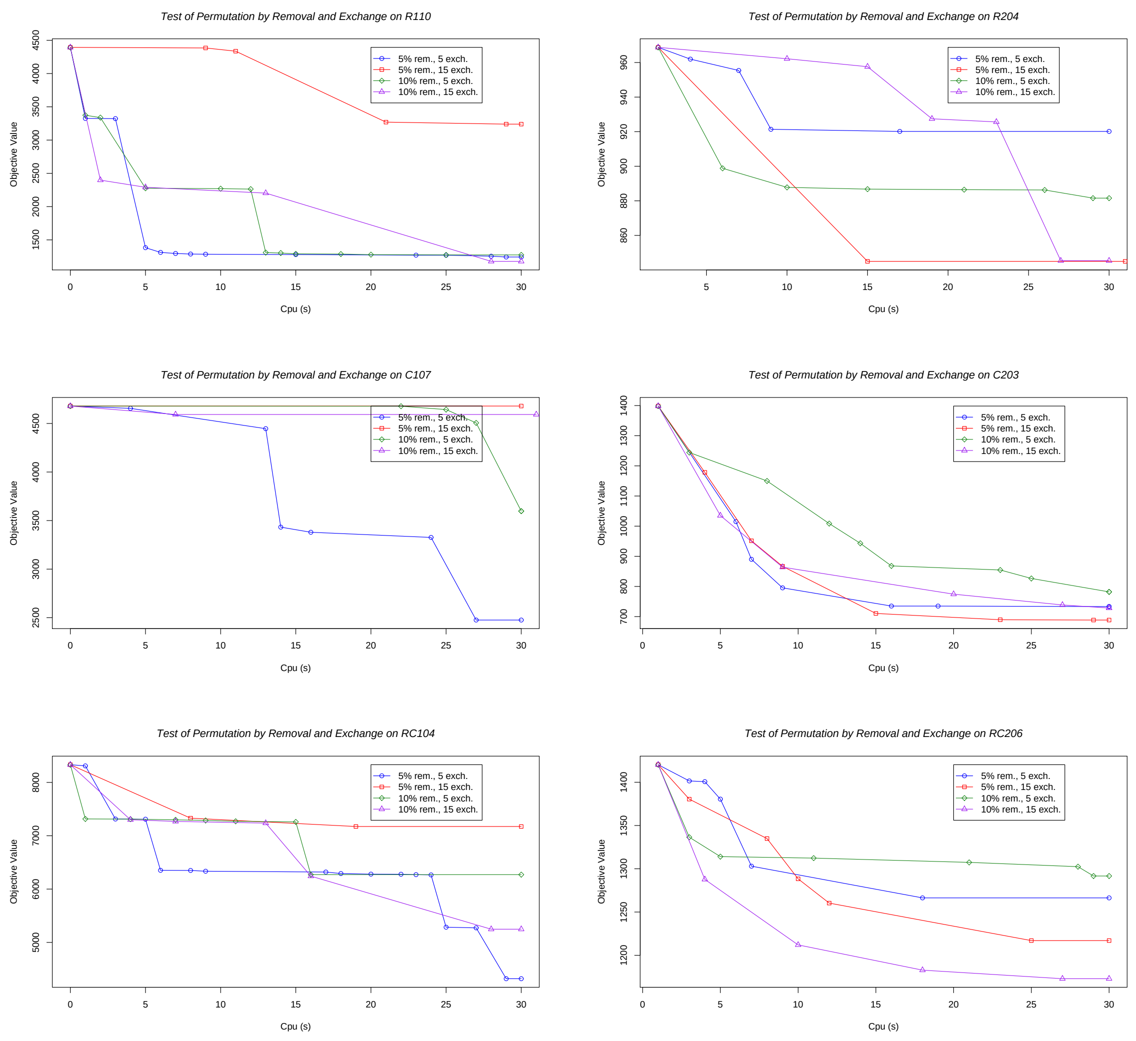


Figure 6.6: Permutation by Removal and Neighbourhood: Graphs of the ILS algorithm running on one Solomon instance of each class. For each instance, four settings were tested. Calling the percentage customers removed a and the number of moves done in the neighbourhood b, the tested settings were $(a = 5, b = 5)$, $(a = 5, b = 15)$, $(a = 10, b = 5)$ and $(a = 10, b = 15)$.

Racing All Permutation Settings

From the previous sections, it is clear that further testing is needed to select the best method and settings of permutation for ILS. Either the settings were not significantly different in the results they produced, or the test base of 6 instances was too small.

To find the best permutation method and setting, a race was set up between all the permutations and settings described above. The race was done on all the Solomon instances. Each run was given 10 seconds, which seems realistic for the purpose of the Online Stochastic Algorithms examined in this thesis. The settings of the race, are the same as described in section 6.1.1.

The output of the race is given in Appendix B.2. Only two candidates were alive at the end of the race, namely permutation by removal of 20% and 25% of the customers. As the setting of 25% found the best solutions most frequently, this is the setting that is used for ILS.

6.5 Comparing ABHC and ILS

In the context of this thesis, offline algorithms are needed for three purposes; when deciding the number of vehicles available for the online instances, to compare the performance of the online stochastic algorithms, and as a sub-procedure for these. For the first two purposes, no there is no time limit for the running time of the algorithms, and hence ABHC seems natural to use, given its good results. The ILS was implemented for the specific purpose of functioning as a fast sub-procedure, that was able to run for as long time as specified while attempting to improve on the solution. It is very easy to modify the ABHC to have a time limit though. Since the ABHC at all times keep track of the best solution found so far, one can simply return this when no more time is available. This makes it usable as a sub-procedure for the online stochastic algorithms. The only problem with ABHC is its natural limit on how long it can improve a solution. But given the good solutions that ABHC is able to find, if it has sufficient time to finish naturally, the solution must be considered sufficiently good.

To compare the ABHC and ILS algorithms in terms of their function as a sub-procedure of the Online Stochastic Algorithms, each was given 10 seconds to run, starting from a solution generated by Impact followed by Ejection Chain. The algorithms were run on the Solomon benchmarks, and the results are compiled in Table 6.7. Surprisingly, ABHC is superior to ILS on every instance. Apparently ABHC is very efficient at finding some very good local minima fast.

During the run of the online algorithms, more and more parts of the route will be fixated, making the solution space smaller. In effect this is similar to scaling down the instance size that the algorithms have to solve. This makes the probability of the ABHC algorithm to finish naturally greater, and hereby find very good solutions.

Based to these results, the ABHC was concluded to be superior as a sub-procedure for the Online Stochastic Algorithms.

Instance	ABHC		ILS		Diff.	
	u	length	u	length	u	length(%)
C101	5	1115.31	5	1221.79	0	9.55
C102	1	1318.63	1	1639.94	0	24.37
C103	0	1095.60	0	1496.13	0	36.56
C104	0	958.62	0	1337.64	0	39.54
C105	3	1253.75	7	1640.60	4	30.86
C106	0	1153.19	5	1798.38	5	55.95
C107	0	1256.02	3	1679.36	3	33.70
C108	0	1165.72	3	1561.14	3	33.92
C109	0	884.00	0	1813.07	0	105.10
C201	0	660.06	0	860.57	0	30.38
C202	0	991.15	4	1336.51	4	34.84
C203	0	620.30	0	1398.44	0	125.45
C204	0	945.84	2	1133.41	2	19.83
C205	1	777.62	1	964.86	0	24.08
C206	0	672.10	0	903.90	0	34.49
C207	0	588.29	1	1038.61	1	76.55
C208	0	678.50	0	1002.93	0	47.82
R101	0	1661.21	0	1753.42	0	5.55
R102	0	1525.29	0	1588.78	0	4.16
R103	0	1291.02	0	1368.40	0	5.99
R104	0	1037.39	2	1146.06	2	10.48
R105	0	1415.37	0	1533.25	0	8.33
R106	1	1295.63	1	1461.00	0	12.76
R107	0	1172.92	3	1252.01	3	6.74
R108	1	984.36	2	1057.04	1	7.38
R109	0	1224.77	2	1460.25	2	19.23
R110	1	1184.45	3	1393.61	2	17.66
R111	1	1172.43	5	1270.16	4	8.34
R112	0	999.77	1	1198.47	1	19.87
R201	0	1246.64	0	1382.79	0	10.92
R202	0	1271.70	0	1383.00	0	8.75
R203	0	1178.71	0	1269.12	0	7.67
R204	0	918.42	0	968.77	0	5.48
R205	0	1036.76	0	1171.15	0	12.96
R206	0	1078.32	0	1159.54	0	7.53
R207	0	922.58	0	960.60	0	4.12
R208	0	788.84	0	885.46	0	12.25
R209	0	971.01	0	1041.95	0	7.31
R210	0	1014.65	0	1150.64	0	13.40
R211	0	834.89	0	933.51	0	11.81
RC101	0	1700.30	3	1855.80	3	9.15
RC102	0	1545.80	0	1911.68	0	23.67
RC103	1	1304.24	2	1345.47	1	3.16
RC104	2	1230.77	7	1332.17	5	8.24
RC105	0	1644.31	2	1843.68	2	12.12
RC106	0	1452.35	0	1499.57	0	3.25
RC107	0	1358.34	3	1408.19	3	3.67
RC108	0	1256.00	3	1445.18	3	15.06
RC201	0	1499.98	0	1687.47	0	12.50
RC202	0	1384.29	0	1511.38	0	9.18
RC203	0	1122.87	0	1219.89	0	8.64
RC204	0	936.22	0	1063.72	0	13.62
RC205	0	1412.61	0	1504.77	0	6.52
RC206	0	1156.15	0	1420.28	0	22.85
RC207	0	1111.83	0	1301.26	0	17.04
RC208	0	929.44	0	1038.34	0	11.72

Table 6.7: Results for Comparison of ABHC and ILS Given a Running Time of 10 CPU Seconds: The instances are the Solomon benchmarks, and the starting solution of the algorithms were produced by a run of Impact followed by a run of Ejection Chain.

7 Oblivious Online Algorithms

When presented with a problem in which the input data becomes known during the execution of the algorithm, an intuitive approach would be to use oblivious online algorithms. Even if stochastic knowledge could be made available, this might be difficult and complicated to acquire as well as writing algorithms to incorporate this knowledge. For this reason, oblivious online algorithms are still used in many real life applications, because of their simplicity and straightforwardness. Note that "oblivious online algorithms" in the context of this thesis describes the class of online algorithms that do not take into account stochastic knowledge of future events.

The oblivious online algorithms are relevant to study in this thesis for two related reasons. First of all, when studying one approach to a problem, it often makes sense to examine the alternatives - in this case oblivious online algorithms. Although other ways of handling the dynamic VRP have been examined, the time frame for this thesis is limited, and the online approach is widely used and seemed the most obvious one to consider. Secondly, to be able to assess how well the Online Stochastic Algorithms perform, we need a base of comparison. By comparing the stochastic algorithms of this paper to online algorithms we get an idea of the value of using stochastic knowledge in online problem solving. Furthermore, a measure can be made ranging from worst (online) to best (offline) solution quality, allowing us to see how well we are able to perform using the Online Stochastic Algorithm.

Three oblivious online algorithms were implemented for this thesis; Nearest Neighbour will be described in section 7.1, Nearest Insertion will be explored in section 7.2 and an algorithm called Local Optimization will be described in section 7.3.

7.1 Nearest Neighbour

Like Van Hentenryck and Bent [2006], the Nearest Neighbour (NN) heuristic was chosen as an online algorithm. This is due to its simplicity and good results for the dynamic vehicle dispatch problem in Larsen et al. [2002].

The basic principle of the algorithm is very simple. When a vehicle has finished service of a customer (is idle), it travels to the nearest neighbouring customer. This continues until they are forced to return to the depot (due to the time horizon). While this seems reasonably efficient in terms of minimizing the route length, the algorithm does not take into account the time windows, and hence the routing plan risks having a fair amount of waiting time. It is worth noting that NN waits as long as possible before making decisions. This allows for new customers to become visible and to be taken into consideration when deciding which customer to serve next. For more details on the algorithm, the reader is referred to Larsen et al. [2002].

7.2 Nearest Insertion

Due to the possible flaws of the NN heuristic, another heuristic was implemented. The approach is a bit different, in that it creates routes using all the available customers, and whenever new customers become available, these are inserted into the routes right away. This is rather different than the NN heuristic, that adds to the route, only when a vehicle is idle. The advantage is, that having already constructed routes, inserting new customers into this will naturally take into account the remaining part of the route, potentially yielding better solutions. At the same time, this means decisions are based on fewer customers than by the NN heuristic, in which customers are only added to a route when the vehicle is idle.

Algorithm 17: Nearest Insertion Heuristic.

```
Data: Online Instance
Result: Routing Plan
1  $\gamma \leftarrow$ initialize empty routes
2  for t $\leftarrow 0$ to h do
3      C $\leftarrow$ new available customer at time t
4      while C $\notin \emptyset \bigvee \exists c \in C, c$ insertable in $\gamma$ do
5          remove most suitable customer from C and insert in $\gamma$ at feasible position with
           minimum increase in $d(\gamma)$
6      end
7      if C $\notin \emptyset$ then
8          add $C$ to set of unserved customers
9      end
10 end
```

An outline of the algorithm, dubbed Nearest Insertion (NI), is given in Algorithm 17. It starts out by initializing each route as an empty tour (start and finish at the depot). Then the main loop is entered (line 2-9). It runs through the entire timespan of the instance (line 2), and whenever new customers become available (line 3), these are all inserted into the routing plan (line 4-6) one at a time, at a position in which they increase the total length of the routing plan by a minimum amount. If it is not possible to insert all the new customers into the routing plan, the remaining customers are assigned to the set of unserved customers.

In both NI and NN, ties are broken by simply using the first one found. In case of NN, this means that if two neighbours are found with the same distance to the idle vehicle, the first discovered is used. For NI, when equally good insertions are found, the first discovered is used.

7.3 Local Optimization: Pool-based Online Algorithm

A third online algorithm named Local Optimization (LO) was also implemented. It is a generalization of the algorithm by Gendreau et al. [1999], and its approach is somewhat closer to that of the Consensus algorithm. It has a partial plan, fixated up to current time t and keeps a pool of solutions based on the known customers which is used to guide the algorithm to good solutions.

The algorithm is outlined in Algorithm 18. It starts by initializing the partial plan y_m as an empty route, and the set R of visible customers. Line 3 calls a procedure for improving on the pool of solutions until it is full (of size p) or no more time is available. This is outlined in Algorithm 19, and will be described below. The main loop (line 4-18) iterates over the entire time horizon. It starts by accepting the new requests R_t, and for each routing plan in the pool, it attempts to insert the customers into it. In the implementation for this thesis, this is done by means of the Ejection Chains algorithm. Following the insertion, a local search is made to improve the newly updated plan. The set R is updated to contain the new customers in line 11.

Algorithm 18: Local Optimization

```
Result: Full Plan $\gamma_m$
Data: Poolsize $p$
1  $\gamma_m \leftarrow$ empty plan
2  $R \leftarrow$ customers available from beginning
3  improveSolutionPool($\Gamma, R, \gamma_m$)
4  for $t \leftarrow 1$ to $h$ do
5      if new requests $R_t$ then
6          foreach $\gamma \in \Gamma$ do
7              Insert $R_t$ in $\gamma$
8              $\gamma \leftarrow$ run local search on $\gamma$
9          end
10     end
11     $R \leftarrow R \bigcup R_m$
12     $P_{id} \leftarrow$ getIdles($\gamma_m$)
13     if $P_{id} \notin \emptyset$ then
14         add customer to $\gamma_m$ based on best plan $\gamma \in \Gamma$
15         $\Gamma \leftarrow$ update and prune plans from $\Gamma$
16     end
17     improveSolutionPool($\Gamma, R, \gamma_m$)
18 end
```

Line 12-16 handles idle vehicles. If a vehicle is idle, the best plan in the pool is consulted, and its choice of customer is used for the partial plan γ_m. The rest of the pool is then pruned for routing plans not conforming to the selected request. In line 17, the procedure for improving on the solution pool is called, and allowed to run for the remaining time.

The `improveSolutionPool` procedure is shown in Algorithm 19. If the pool is not full, new routing plans are generated based on the partial plan γ_m and the unserved customers $R \backslash cust(\gamma_m)$. This continues to the pool is full or no more time is available. If time is still available and the pool is full, the routing plans of the pool are optimized until no more time is available.

The idea in keeping a pool of routing plans, even though no sampled customers are used, is to attempt to accommodate the unknown customers. The more plans available, the greater the chance is for having a plan that is suitable for inserting the new requests.

Algorithm 19: Improve Solution Pool

```
Data: Pool of solutions ($\Gamma$), visible customers $R$, partial plan $\gamma_m$.
while time is available $\wedge \Gamma$ is not full do
    $\Gamma \leftarrow$ generate solutions based on $R$ and $\gamma_m$;
end
while time is available do
    foreach $\gamma \in \Gamma$ do
        optimize on $\gamma$;
    end
end
```

The algorithm that is improving and generating solutions in `improveSolutionPool`, must not be deterministic as this would lead to a pool of identical solutions. Therefore the ABHC cannot be used for this, and instead ILS was used.

As can be seen, the LO algorithm is in many ways similar to the Consensus algorithm, but with two important differences. The Consensus algorithm bases its pool on sampled customers, whereas LO only takes known customers into account. Furthermore, Consensus bases its choice of customers for idle vehicle on a consensus of all the plans in the pool, whereas LO bases it on the best plan in the pool.

7.3.1 Tuning the Pool Size

Besides finding an offline solver, which has been done in section 6, the only thing to tune in the LO algorithm is the pool size. For this, a race was set up between pool sizes of 1, 5 and 10. Note that a pool size of 1 is similar to have an offline solver improving on the solution at all times available, and adding customers from it to the partial plan only when needed (ie. when a vehicle is idle).

The race settings were the same as those used in section 6.1.1. The algorithm had 1 CPU minute before the time horizon started, and then 10 CPU minutes for the entire run. It was run on all the Solomon benchmarks, and for each instance a random class was chosen to make the problem online (see section 5.4). This should test the algorithm on a variety of different instance types and on different amounts of dynamic customers.

The output of the race is printed in Appendix B.3. A pool size of 5 gave the best results, but neither of the settings was superior enough to discard any of the other two candidates. For the remainder of this thesis a pool size of 5 will be used.

8 Using Historical Knowledge

For this thesis, exact knowledge of the stochastic distribution of customers is available for the instances created via the instance template described in section 5.2. But often in real life applications this is far from realistic. In most cases, no models for the distribution of the customers are available, and if it is, it is likely to be either outdated, or too imprecise.

For some online problems, the online algorithms are run continuously or for very long periods of time, eg. packet scheduling. For these types of problems, one can attempt to learn the distribution online using for example Hidden Markov Models. Here, subsequences of the input might reveal information of the distribution and may be used to infer the state of distribution or train the model. Another approach is to look at the past x time steps, and derive the probability that a specific request arrives in any of the subsequent time steps. This latter technique is called historical average. While these techniques are usable for some applications of Online Stochastic Algorithms, they are not suitable for the online stochastic vehicle routing problem dealt with in this thesis. For more information on these, the reader is referred to Bent and Van Hentenryck [2005].

The vehicle routing problem is a great deal more complicated than for example packet scheduling, and stochastic distribution of the requests would seems infeasible to model via HMMs or the alike. Historic average lacks the ability to capture the structural properties of the distributions, so instead the approach of historical sampling is taken. This is described in the following section.

8.1 Historical Sampling

As mentioned above, the VRP is significantly more complicated than a problem like packet scheduling — consequently it is very hard to model the stochastic distribution of its variables using a HMM. In principle, historical averaging could be used, but it would not be able to capture the structural properties of the distributions, such as customers arriving late in the day, etc.. Rather, the approach of historical sampling seems very suitable for capturing the properties of the VRPTW [Bent and Van Hentenryck, 2005].

The idea of historical sampling is to take entire instances from the past and use these as samples. The algorithm for historical sampling for the VRPTW is given in Algorithm 20. It takes a pool of past instances along with the current time t, the set of visible real customers that have not yet been served by the current plan, as well as the current plan y.

In line 1, a plan is selected at random from the pool of past instances. Line 2 prunes the instance for all the customers that have earliest service time before time t (ie. are in the past) or not reachable from the current plan γ before their latest time of service. To take the real requests that are already visible, but have not been served yet, into account, this amount of customers

($|R|$) are removed from the set S. This gives the expected amount of customers that still remains to become visible. The set S of customers is finally returned as the sampled set. With this approach, the time of day customers appear is taken into account and a realistic sample of the remaining customers is obtained.

```
Algorithm 20: Historical Sampling
  Data: Pool of historical instances ($\Upsilon$), current time ($t$), set of unrouted real customers $R$,
        current plan $\gamma$.
  Result: Set of sampled customers $S$
1 $\upsilon_s \leftarrow$ select random instance from $\Upsilon$;
2 $S \leftarrow$ customers $c \in \upsilon_s$ with $e(c) > t$, and reachable by $\gamma$ before $l(c)$;
3 for $i \leftarrow 1$ to $|R|$ do
4     $S \leftarrow S-$ random customer $s \in S$;
5 end
6 return $S$;
```

There are several advantages of historical sampling. First of all, this data will be available, or can be made available very easily for most applications. It is very simple to implement, but yet is able to capture structural properties of the problem, like customers in certain areas appearing late, specific times of day that are particularly busy, etc..

A potential extension to historical sampling is to split the historical instances into different classes. As an example, a taxi service will most likely have different distributions and amounts of requests in weekends and weekdays. This can be modeled by splitting the pool of historical instances into two classes; weekend and weekdays. If the day currently being solved is a weekend, only the pool of weekend instances is sampled. Of course, this could be further split into holidays, special events, etc..

An interesting consideration in connection in relation to historical is how many historical instances need to be available before the historical sampling is able to give good results. This is tested in section 9.5 along with a comparison to exact sampling.

8.2 Exact Sampling

For this thesis, an exact probabilistic model is available in form of the instance template described in section 5.2. As mentioned, this is not realistic in practical applications, but having it available for this thesis serves as a basis for assessing how well the historical sampling performs, both as a mean of comparison and by being able to create “historical instances”. This can be done by sampling instances using the instance template and adding this to the Υ pool. This is repeated until the pool has the desired size.

9 Solving the Online Stochastic Vehicle Routing Problem

After having described the model and algorithms for the online stochastic vehicle routing, we can now look at their performance. In this chapter different aspects of the algorithms will be examined, to see how they perform when presented with various difficulties. In the end, a comparison will be made between the oblivious online algorithms, the Online Stochastic Algorithms and the offline algorithm ABHC. This gives us a scale to evaluate the performance of the Online Stochastic Algorithm on, and furthermore conclude if there is any gain in using stochastic information to guide the algorithm.

9.1 Test setup

To evaluate the Online Stochastic Algorithms, the benchmark instances described in section 5.3 are used. A real life time horizon of 30 minutes is given for the solving of an instance, plus 5 minutes for the initial generation of plans, based on customers that are available from time $t = 0$. As previously described, the 30 minutes time horizon allows 3 seconds per time step for the 180 instances, and 10 seconds for the 600 instances. This, of course, is much less than the algorithms would have in most practical use, where the real life time horizon would often be 8 hours. To compensate for this, as described in section 5.3, the instances were scaled down, so solving one of the generated benchmarks in the 30 minutes available would be somewhat similar to solving a realistic instance on an 8-hour time horizon. If nothing else is described, the instances are made online using the online class 3 of Table 5.2, described in section 5.4, page 38.

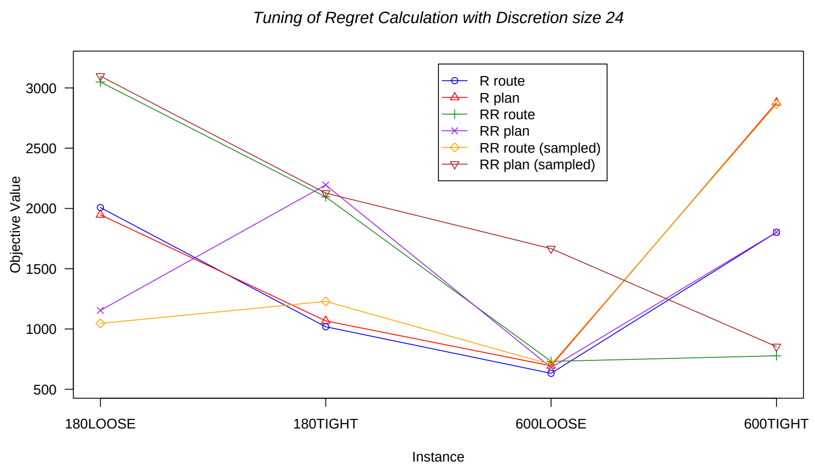
For all the Online Stochastic Algorithms, the initial solutions ($t < 0$) are generated using the Impact algorithm, followed by a run of Ejection Chains and ABHC with Best Fit. From this point, the solving of sampled instances is done by Ejection Chains followed by the ABHC with Best Fit. If nothing else is specified, the ABHC has a maximum of 10 seconds to solve an instance.

The testing was done on the computers of IMADA, University of Southern Denmark. These are Intel Core2 6300 (1.86GHz) processors with 2GB of memory.

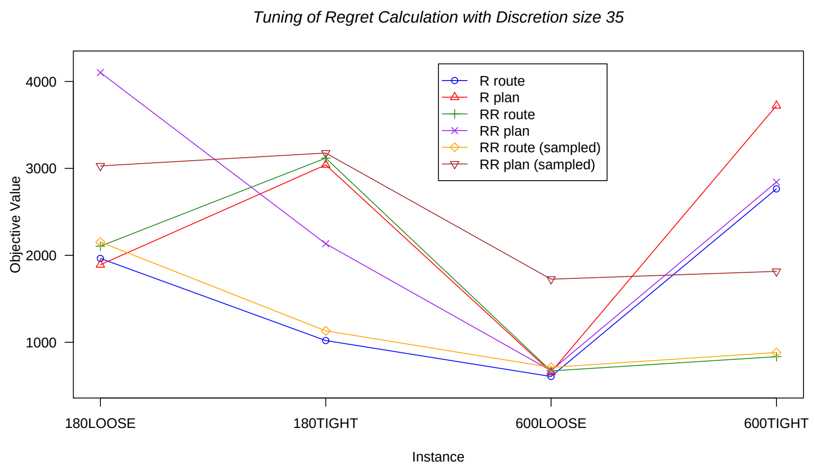
All the times measured are CPU-time.

9.2 Regret Customers

As described in section 4.4, it was not clear from the papers of Van Hentenryck and Bent for which customers regret should be calculated — for the current route of the customer or the entire



(a) Results of the Regret algorithm having different strategies for calculation of regret. Run with a discretization size of 24



(b) Results of the Regret algorithm having different strategies for calculation of regret. Run with a discretization size of 35.

Figure 9.1: Graph of the Solution Quality of the Regret Algorithm, Run with Different Types of Regret Calculations: These differ in which customers the regret is calculated for. The figures show results for the four generated benchmark instances. Results are connected by lines to make the performance of the algorithms more clear for the reader. Figure 9.1(a) shows results for the algorithm, when the discretization size is set to 24, and Figure 9.1(b) show for discretization size of 35.

routing plan. Moreover, when using the Relocation strategy, it needs to be considered whether regret should be calculated for sampled customers also.

To decide this, different settings of the Regret algorithm were compared. For regular Regret, the regret was calculated for real customers in the route and for real customers in the entire plan.

Regret with Relocation was tested with regret-calculation for both all and real-only customers in the route as well the plan. The different settings were run on the four generated benchmark instances described in section 5.3. This was done with a discretization of map of 24 and 35. This means that the map is split into 24 and 35 units for each axis, yielding 576 and 1225 discrete areas (see section 4.6.1). The results are presented in Figure 9.1. In the legends, *RR* denotes Regret with Relocation strategy and *(sampled)* means that regret is also calculated for sampled customers. The cost of having an unassigned customer in a routing plan is set to 1000. The exact solution values are given in text in Appendix A.2.

Looking at the standard Regret algorithm (without Relocation), calculating regret for only the route (marked in blue) outperforms normal regret calculation for the plan in all instances but 180LOOSE.

Turning to the Regret algorithm using Relocation (RR), the results are not as clear. As can be seen, it is preferable to calculate the regret from the route only, but whether to include sampled customers in this calculation is unclear, and seems to be dependent on discretization size. The choice was made to include sampled customers in the regret calculations. This seems natural since Regret with Relocation considers real and sampled customers equal, and it makes sense to do this in the calculation of regret also.

9.3 Size of Discretization

Section 4.6.1 describes the idea of discretization of the map. As opposed to all positions of the map being unique, ranges of positions are grouped into areas and the online stochastic algorithms consider these as one unit when calculating consensus, etc.. Since only discrete positions are used in this thesis, on a map of 70x70 units as the generated benchmarks, using no discretization would be similar to discretizing the map with 70x70 areas containing one discrete position.

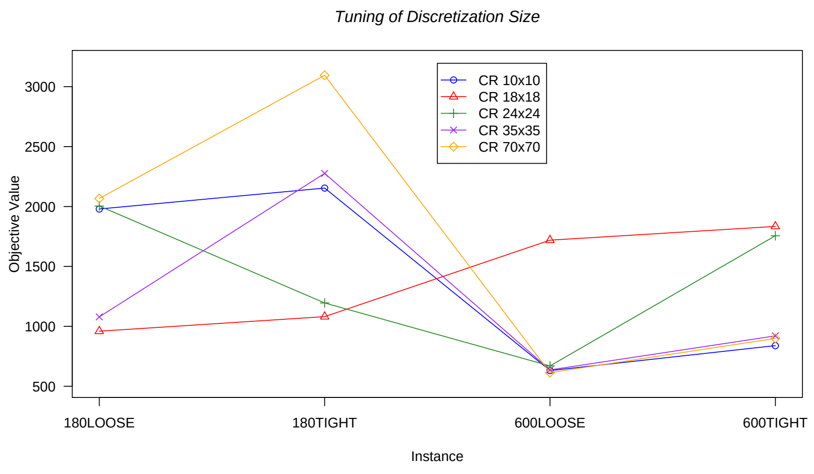
Discretization is only examined for the Relocation variants for Consensus and Regret, since only these allow consensus for sampled customers and hence are the only ones it make sense to use discretization for.

For examining the effect of discretization, runs were made with 10, 18, 24, 35 and 70 splits per axis. This gives areas of 7^2, 4^2, 3^2, 2^2 and 1 discrete positions, respectively. Each setting was run once on each of the four generated benchmark instances. The results are displayed in the graphs of Figure 9.2 and the numeric results are given in Appendix A.3.

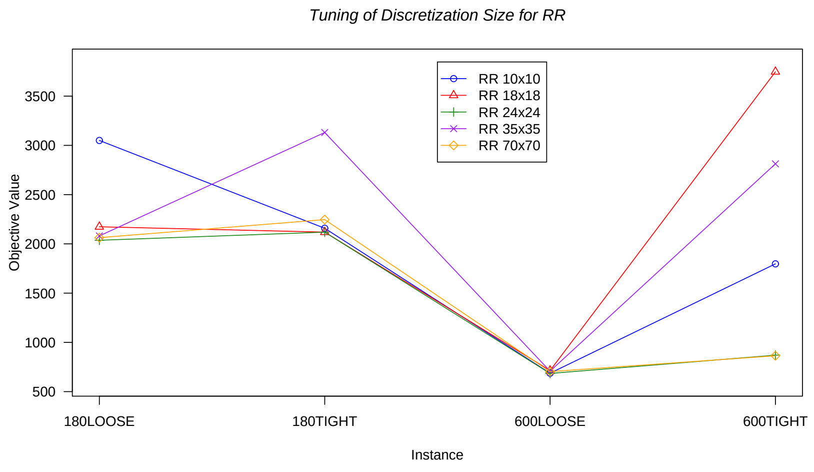
Although the Regret algorithm using Relocation get very similar results with a 70x70 and 24x24 discretization, the setting of 24x24, ie. 9 positions per area, performs best in all cases but one (600TIGHT) in which 70x70 performs best. The regret was calculated for the route including sampled customers.

For CR the results are less clear. A discretization of 10 and 18 finds good results for the 180 and 600 instances, respectively. The explanation could be the structural properties of the two classes. Recall that the two 180 instances have the same positions for customers, and so do the two 600 instances. Possibly these two positionings are fit for a discretization of 10 and 18, respectively. Even though a setting of 35 does not give the best results in any run, it seems to be more stable than the others - it might be that it is not as affected of the structure of the customers.

It is difficult to asses what setting of discretization is best, based on only 4 instances, that



(a) The effect of different discretizations for the Consensus algorithm using Relocation.



(b) The effect of different discretizations for the Regret algorithm using Relocation.

Figure 9.2: Results of Different Degrees of Discretization: The graphs show the solution quality of the Regret and Consensus algorithms with Relocate run on the generated benchmarks with different degrees of discretization. These are run on the four generated benchmarks. The numbers in the legend, describe the number of areas the map is split into. The weight of unrouted customers are set to 1000.

only have 2 different positionings of the customers. Furthermore, a map of a small size like 70x70 units, in which the customers can only be placed in discrete (integer) positions, the effect of discretization might not be as clear. The discretization size is likely to depend on the size of the map. A map of size 500x500 with discrete, or even continuous positions, containing 100 customers could lead to a different setting and give a clearer picture of the effect of discretization. In any case, it is obvious from the results that discretization *does* have an effect compared to not discretizing, and furthermore this effect seems positive.

It would definitely be interesting to compare the effects of discretization on a bigger map, and possibly with continuous positioning, or on more instances, to get a more clear image of its effect and a better idea of what degree of discretization is suitable for different map properties.

9.4 Sampled Solutions: Quality vs. Quantity

For the Online Stochastic Algorithms described in this thesis, a very important part is the offline algorithms. While it is obviously desirable to have an efficient algorithm finding high quality solutions, the balance of the quality and quantity of sampled solutions is not as clear.

If the offline algorithm is given short time to run, this will allow the Online Stochastic Algorithm to generate and solve more instances. When many sampled instances are solved, a better statistical basis will be available for the online stochastic algorithm. On the other hand, the Online Stochastic Algorithms base their decisions on the routing plans found by the offline algorithms, and if these are of low quality, this might be reflected on the solution. Solving only few sampled instances but doing so thoroughly, will let the Online Stochastic Algorithm base its decision on good routing plans, but few of these mean that the statistical basis for the decisions is not as good.

To examine this, a comparison of the Online Stochastic Algorithms run with different settings of the offline algorithms is needed. Since the real life time horizon is set to 30 CPU-minutes (excluding 5 CPU-minutes of sampling and solving instances with the customers known at time 0), 10 CPU-seconds are available per time step for the 180 instances, and only 3 CPU-seconds for the 600 instances. For this reason, the testing of quality and quantity of sampled solutions was made only on the 180 instances, since this allowed the offline algorithms a running time of up to 10 seconds. Five different offline settings were tested. A simple Best Fit was used as the fastest offline algorithm. Besides this, 4 different settings of ABHC were tested, with 1.5, 3, 6, and 10 seconds of running time, respectively.

9.4.1 Results

The Consensus(C) and Regret(R) algorithms were run with a discretization size of 70x70. Their Relocation variants CR and RR, were run with discretization sizes of 35x35 and 24x24, respectively.

The results of the runs are compiled in Table 9.1. In terms of C and R, it is clear that good quality of solutions is to prefer over quantity. For both the loose and tight versions of the 180 instances, the best results were found with a setting of 10 seconds for the ABHC.

The results for the algorithms using relocate are not as clear. In two cases, ABHC with 1.5 second of running time performs best, while Best Fit gives the best result for RR on 180LOOSE and ABHC with 10 seconds finds the best solution for CR the 180TIGHT instance. To examine this further, the comparison was reproduced for the 600 instances. Since these allow no more than 3 seconds of running time per time step, only the settings of Best Fit and ABHC with 1.5 and 3 seconds running time could be tested. In terms of C and R it seemed clear from Table 9.1 that it was desirable to give them a long running time, so there was no need to include them in

180LOOSE					
	Best Fit	**ABHC 1.5**	**ABHC 3**	**ABHC 6**	**ABHC 10**
C	3 903.98	2 846.90	1 928.33	1 888.67	**0 918.12**
R	4 895.18	1 942.76	1 1028.75	2 953.57	**1 921.83**
CR	2 1144.82	**1 1060.23**	2 1100.63	1 1086.77	2 1086.71
RR	**1 1173.39**	2 1077.85	2 1114.12	2 1136.24	2 1138.82
180TIGHT					
	Best Fit	**ABHC 1.5**	**ABHC 3**	**ABHC 6**	**ABHC 10**
C	3 1004.77	0 1030.99	0 1018.64	0 1027.01	**0 1018.20**
R	3 1026.82	0 1054.86	1 1014.16	1 1014.31	**0 1053.65**
CR	3 1117.41	0 1211.64	0 1128.98	2 1157.60	**0 1111.30**
RR	3 1142.40	**0 1152.99**	0 1223.40	2 1196.63	1 1200.39

Table 9.1: Results of the Online Stochastic Algorithms with Different Settings of Offline Algorithm: This table displays the results of the regular Consensus (C) and Regret (R) algorithms and Consensus with Relocation (CR) as well as Regret with Relocation (RR) run with different settings of offline algorithms. The 1.5, 3, 6 and 10 denotes the maximum allowed running time of the ABHC in CPU-seconds. Obviously all the time available in a time step is used to generate and solve instances. Consequently ABHC 1.5 is likely to be run many more times than ABHC 10. In each cell, the left number denotes unserved customers, while the right number is the length of the routing plan. Best results are marked with bold types. The displayed results are for the instances 180LOOSE (top) and 180TIGHT (bottom)

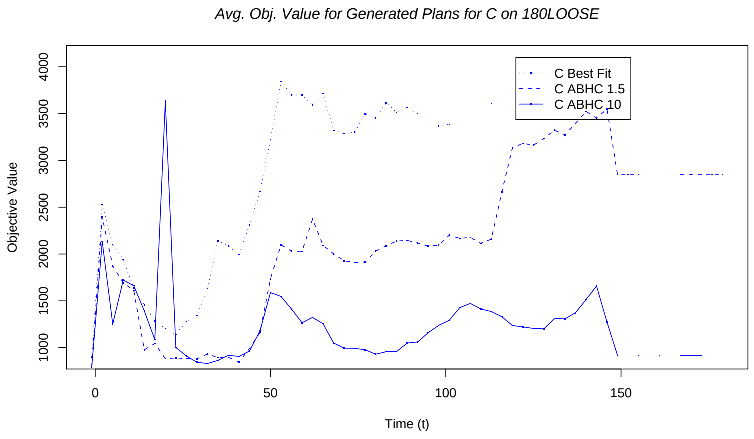
the test on the 600 instances. For the Relocation, in 3 of 4 cases, the best results were found when the offline algorithms had a short running time, so it makes sense to examine this further on the 600 instances, to make it more clear what setting is preferable.

The results for the 600 instances are presented in Table 9.2. As in the case of the 180 instances, for the loose version, RR finds the best solution using Best Fit followed by ABHC 1.5. In all other cases, ABHC performs best with a setting of 1.5.

A possible reason for the difference in choice of offline algorithm for the regular C and R algorithms, as opposed to their Relocation counterparts, could be their dependency on sampled customers. As mentioned above, more sampled instances gives a better representation of this distribution, and hence allows Relocation to serve more relevant samples. The regular C and R only serves real customers, and while sampled customers affects their decisions, it comes down to which real customer is next on a route. Therefore, C and R might be more tolerant of unrepresentative samples.

600LOOSE			
	Best Fit	**ABHC 1.5**	**ABHC 3**
CR	0 721.94	**0 690.88**	0 759.41
RR	**0 747.85**	0 794.26	2 710.81
600TIGHT			
	Best Fit	**ABHC 1.5**	**ABHC 3**
CR	3 926.76	**0 853.22**	0 892.01
RR	2 849.37	**1 821.89**	1 847.47

Table 9.2: Table Displaying the 600 Instance Counterpart of Table 9.1: Best results are marked with bold types.



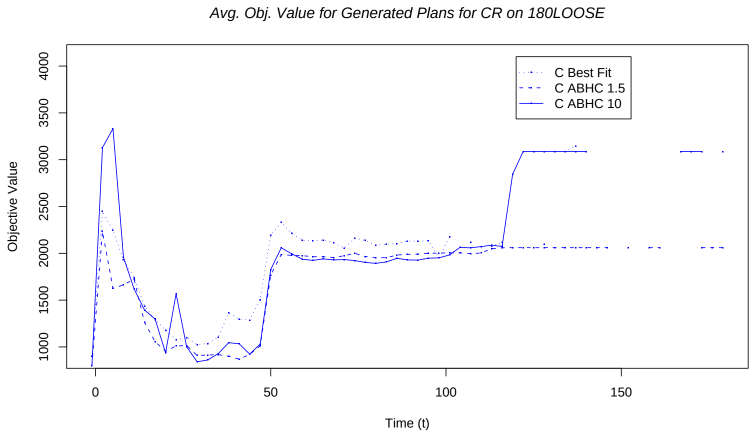
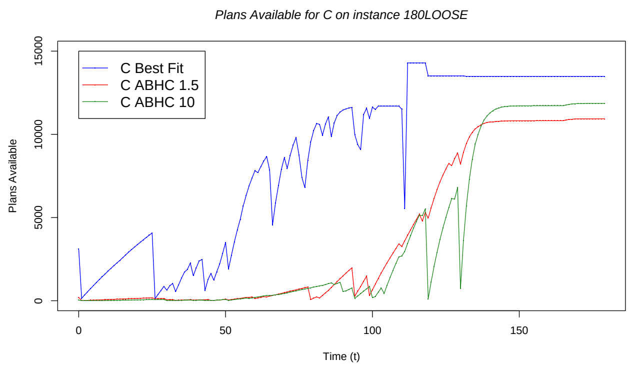


Figure 9.3: Solution Quality for Sampled Solutions: Graphs showing the average objective value of the sampled and solved instances over the time horizon. For every three time step, an average of the objective functions of the plans generated and solved during these time steps is shown. This was done to smooth the graph out. The results are of runs of the C and CR algorithms run on the 180LOOSE instances. Unassigned customers contribute with 1000 to the objective function.

9.4.2 Further Examination

When examining the effect of quality and quantity, it seems relevant to look at the actual solution qualities found by the different algorithms over the course of execution. An example of these are displayed in Figure 9.3.

For the C algorithm, the solutions generated by Best Fit (Descent) are inferior through the entire run. The two ABHC settings are more equal until around time 50, when ABHC 10 generally manages to insert 1 more customer customer into the route. As can be seen, Best Fit is



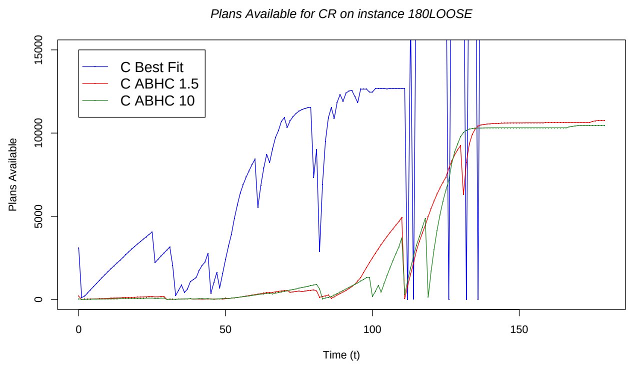


Figure 9.4: Quantity of Sampled Solutions: Graphs showing the number of plans available over the course of time for instances 180. Unassigned customers contribute with 1000 to the objective function.

generally more smooth than ABHC 1.5 which is in turn more smooth than ABHC 10, especially in the beginning of the timespan. This is due to the many more plans generated by the Best Fit algorithm.

The picture is not as clear for the CR algorithm, where the three offline algorithms are more equal. At around time 120, ABHC 10 and Best Fit generally fails in serving 1 more customer than ABHC 1.5.

It should be mentioned that when running ABHC 10, for example, the algorithm will not necessarily have a full 10 seconds to run. Sometimes, only little time will be left over from calculation of Consensus, pruning, etc.. This could explain the very bad solutions that seems to be found in the early time steps from ABHC. Since, in this period, there are only time for around one optimization, in case of ABHC 10, a very bad solution will have a great effect on the average objective value. As we move forward in time, a greater part of the plans will be locked, and therefore the ABHC will often be done withing the 10 seconds. Therefore, the fluctuations are not as big late in the graph as in the beginning.

Besides the quality of the solutions, the quantity of sampled and solved instances might have an effect on the solution. The results are displayed in Figure 9.4. As could be expected the pool of plans available for Best Fit is generally larger than that of ABHC 1.5, which is in turn generally larger than that of ABHC 10. At many points during the run of the algorithm, the plans available suddenly becomes smaller. This is due to decisions being made for an idle vehicle. When this happens, the plans that do not agree with this decision are pruned.

There are a few curiosities in connection to the graphs of Figure 9.4. First of all, it seems like the generation of plans stops at around 130 (earlier in the case of Best Fit). There could be several reasons for this. At this point, all or most of the routing plan will be fixated, and so the freedom for the offline heuristics become very small or non-present. Therefore generating and "solving" an instance, is as easy as sampling it from the instance template. This might allow for generation of many plans (10 thousand+). A possible explanation that no more routes are generated, is that, at every time step, the sampled routes have to be fixated according to the current time t. While this is generally very fast, having to do this for 15000 plans might be time consuming, and hence allow little or no time for generation of new plans. Whatever reason, one would have expected the pool size to grow increasingly towards the end. While this is unfortunate, it should not have any significant effect on the solution quality since the basis for making decisions should be sufficient with 10.000+ plans.

The other curiosity is the behavior of Best Fit, when looking at the CR algorithm. It takes some dives from 20.000+ plans, down to as few as 1 plan towards the end of the timespan. The same pattern emerged for RR, while not for R. Note that vehicles do not move to depot until necessary. Therefore, the Consensus of a vehicle could be on a sampled customer, even though all but one plans agrees on the depot. This could be a possible explanation for the dramatic pruning. After a great dive in the quantity of plans, the pool is quickly filled due to a great part of the plan being fixated, resulting in very fast generation and solving of plans.

9.5 Historical Sampling

As described in section 8.1, exact stochastic knowledge of the instance is rarely available in real life applications. For this reason, alternative methods have to be used. One approach would be to use historical sampling, which is what is tested in this section. In real life this can be obtained simply by using historical instances (data from previously solved instances). In context of this thesis, "historical data" is generated by sampling full instances from the relevant instance template x times, where x denotes the number of historical instances that should be available.

The greater the amount of historical data available is, the closer one would expect to come to the actual exact stochastic properties of the instance. This means, ideally, that the more historical data that is available, the better solutions the algorithms should be able to find.

For testing the historical sampling, the algorithms were run with pools of 1, 10, 100, and 1000 historical instances available. Furthermore, results using the exact stochastic model (instance template) were found as a base of comparison.

The results are shown in Figures 9.5-9.8. The exact numbers of the reported results are given in Appendix A.4. Overall there is a pattern of better solutions being found when more historical data is available, but not as consistently as one would have expected. Especially the two Regret algorithms behave oddly. At very small amounts of historical data (1 and to some degree, 10), the quality of the solution found will naturally be very dependent on whether the historical instance happens to be very similar to the instance being solved, in terms of time windows, positions, etc.. If this is the case, good results will be found, and oppositely, if the historical instances are far from the instance being solved bad solution will be found. When reaching a pool size of 100 or more, the probability of getting an “unlucky” or “lucky” pool of samples becomes negligible.

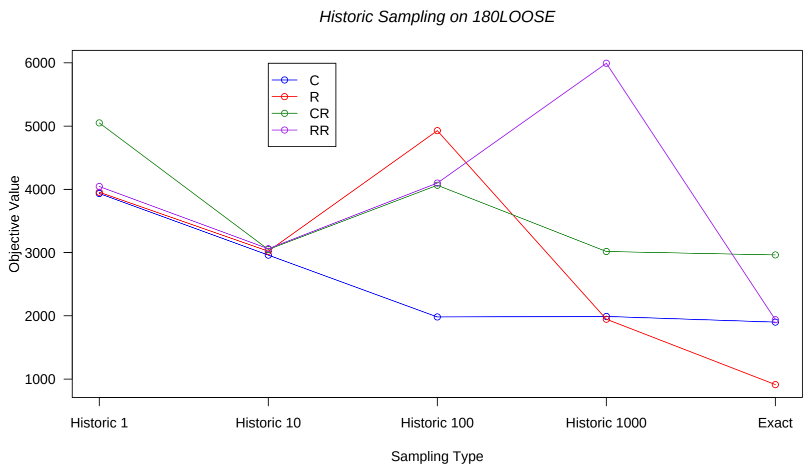


Figure 9.5: Results on 180LOOSE Using Historical Sampling: The figure shows the results of the Online Stochastic Algorithms using different historical sampling on different amounts of historical data. Results of precise sampling is also displayed for comparison.

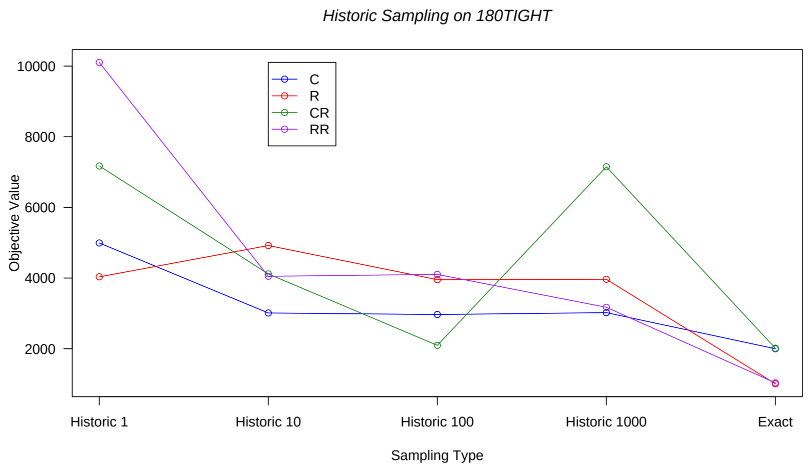


Figure 9.6: Results on 180TIGHT Using Historical Sampling: The figure shows the results of the Online Stochastic Algorithms using different historical sampling on different amounts of historical data. Results of precise sampling is also displayed for comparison.

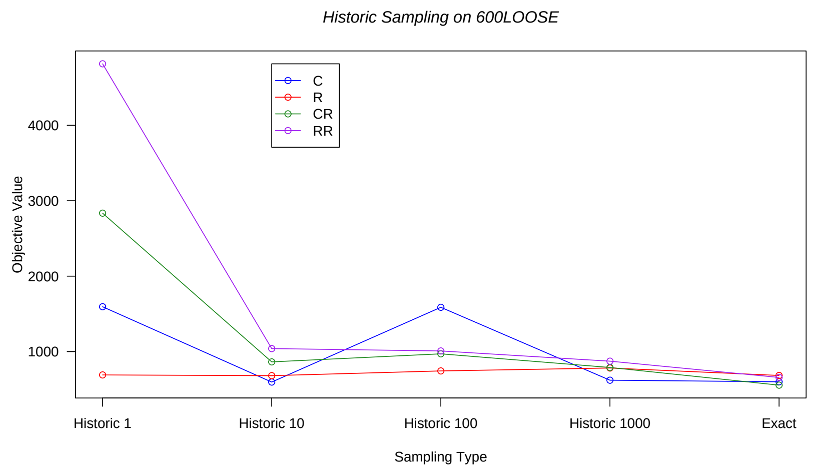


Figure 9.7: Results on 600LOOSE Using Historical Sampling: The figure shows the results of the Online Stochastic Algorithms using different historical sampling on different amounts of historical data. Results of precise sampling is also displayed for comparison.

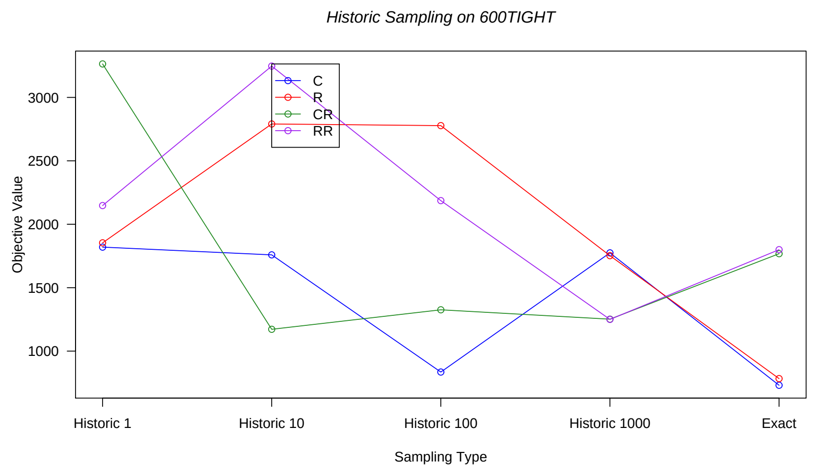


Figure 9.8: Results on 600TIGHT Using Historical Sampling: The figure shows the results of the Online Stochastic Algorithms using different historical sampling on different amounts of historical data. Results of precise sampling is also displayed for comparison.

If more time had been available for examining the properties of historical sampling, the algorithms could have been run multiple times with other "historical data" (different pools of 1, 10, 100 and 1000). This would help draw a clearer picture of the effects of historical sampling.

In any case, from the results displayed in the figures, there is a tendency of finding improving solutions the more historic instances are available, and in the most cases, having eg. a basis of 100 historical instances results in better solutions than having a single historical instance available.

9.6 Comparing with Offline and Online Algorithms

This section compares the Online Stochastic Algorithms with the oblivious online algorithms and the offline solution. This is done in the 5 different online classes of instances, described in Table 5.2 (section 5.4, page 38). The idea is to evaluate how the different Online Stochastic Algorithms perform compared to each other, the online algorithms and the offline solution, and furthermore to examine the influence of the number of dynamic customers in the instance solved.

The offline solutions are produced by the ABHC algorithm. The Consensus and Regret algorithms without Relocation were run "without" discretization (that is, a discretization of 70x70 squares, giving one discrete area one position, which is still a very rough discretization compared to continuous positionings of customers) and the ABHC at a 10 CPU-seconds time limit. Their Relocation counterparts were run with a discretization of 24x24 and 35x35, respectively, and both used ABHC with 1.5 CPU-second running time as offline search procedure. For all the Online Stochastic Algorithms, exact sampling was used (that is, sampling directly from the instance template, as opposed to using historic sampling).

The online algorithms examined are the NN, NI and LO algorithms described in Chapter 7. The NN algorithm was also used by Van Hentenryck and Bent, as well as an algorithm very similar to the LO algorithm implemented for this thesis.

The number of vehicles available for the instances were found by a run of an offline ABHC using the objective $w_1(\gamma)$ (see equation (2.7), page 8), in which the number of vehicles is minimized. This means, that for each instance, one vehicle less than reported in Table 5.1, page 36 is available.

Since Consensus, Regret and their Relocation counterparts are dependent on samples, which are stochastic elements, each algorithm is run 7 times for each instance and setting, to find their average performance. The results reported in this chapter is the mean value of their solution value. While this gives a clearer picture of their average performance, some details are lost in the algorithms stability, eg. one algorithm could be finding very good solutions one third of the time, while another could be stable at finding average solutions.

The results are shown in Figures 9.9 and 9.10. In the reported objective functions, unassigned customers contribute with 1000 to the objective function. The exact values of the solutions are given in Appendix A.5.

Oblivious Online Algorithms Looking at the oblivious online algorithms, NN is in general finding better solutions than LO. These are both inferior to the NI, the Online Stochastic

Algorithms and offline algorithms. In general NI is performing very well, and gives the best result following the offline algorithm in one case.

In general, the LO algorithm produces bad solutions. In all cases but one (online class 5 for the 180TIGHT instance), it performs worst of all the algorithms examined. In particular, relative to the other algorithms examined, it seems to be performing bad for the loose instances, ie. instances with large time windows. Furthermore, the LO algorithm seems very influenced by the number of dynamic customers. This also makes sense, in that the algorithm bases its choices only on the known customers, and having fewer of these available is likely to result in worse solutions.

As mentioned Nearest Neighbour (NN) performs better than LO, but in general manages to place at least one less customer in the routing plans, compared to the other algorithms. It is not as affected by the degree of dynamic as LO, but this due to the algorithm not being as dependent on the number of visible customers as LO.

As opposed to NN and LO, the Nearest Insertion algorithm performs surprisingly well and is competitive with the Online Stochastic Algorithms. In a single case, it finds the best solution (class 5 for instance 600TIGHT) following the offline algorithm. Relative to the Online Stochastic Algorithms it seems to be better at handling short routes (few customer per route) than long routes (many customers per route). Considering only the oblivious online algorithms, this is by far the best.

Online Stochastic Algorithms Unlike the case of the oblivious online algorithms, there is no obvious pattern in which Online Stochastic Algorithm is best - it varies from instance to instance and from online class to online class.

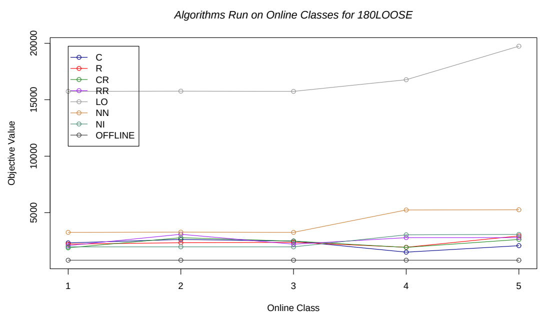
Looking at the Consensus and Regret algorithms not using Relocation (C and R), the Consensus algorithm is generally superior to Regret, with the exception of having low amounts of dynamic customers (online class 1 and 2) on the instances with short routes (the 180 instances).

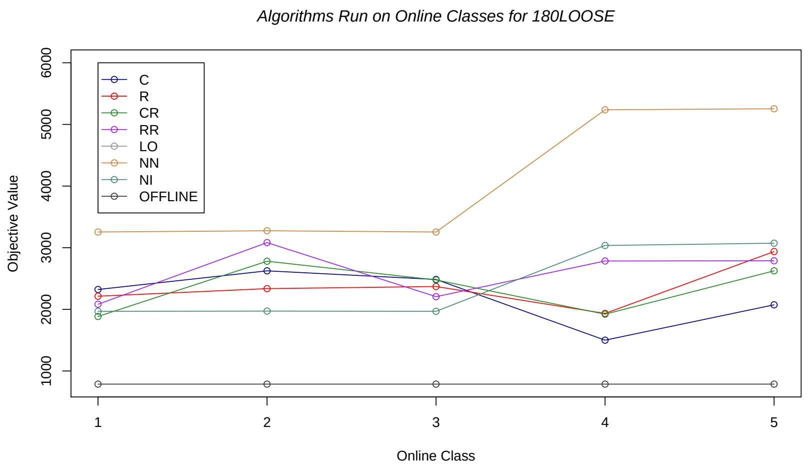
The Relocation counterparts of Consensus and Regret (CR and RR) give less obvious results. Disregarding 180TIGHT, CR generally performs best. For 180TIGHT, RR is superior for all online classes. Besides this, there seems to be no generel pattern predicting which algorithm performs better than the other.

In most cases, not using Relocation seems to be preferable, but this is definitely not consequent. As an example, in the online class 5 for both the 600 instances, the R and C algorithms perform worse than their Relocation counterparts.

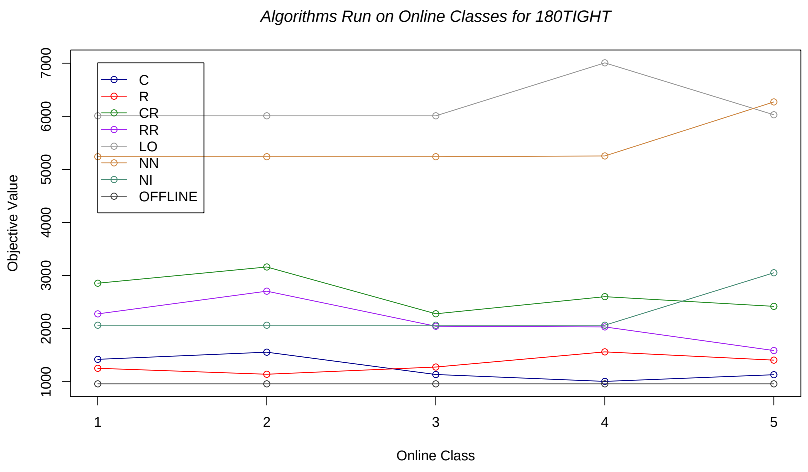
These results are not consistent with those of Van Hentenryck and Bent. In their book, the reported results are that in general R is to prefer over C, and RR is preferable over CR. Furthermore only with very low amounts of dynamic customers are the C and R algorithms superior to their Relocation counterparts. When a great part of the instance is dynamic, the Relocation variants are much superior to C and R. There can be several reasons for this inconsistency. The instances the algorithms were tested on are not the same, and maybe more importantly, the precision of the stochastic knowledge might be different. If Van Hentenryck and Bent eg. have very exact stochastic knowledge and very high probability for “guessing” correctly when sampling customers, this could greatly affect the efficiency of the algorithm, or more precisely, the feasibility of the Relocation algorithms. Besides this, differences in perception of the algorithm details, and consequently implementation, could obviously be a reason for the differences in result.

Comparing the Online Stochastic Algorithms to NI, the Consensus algorithm without Relocation is preferable over NI in most cases. For the online classes 1-3, NI performs best on the 180LOOSE instance, as well as on the online class 5 on both the 600 instances. In all other cases the C algorithm is superior.



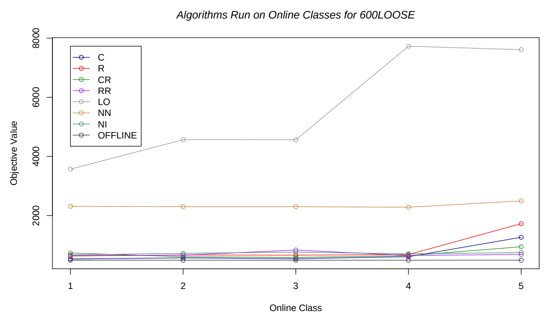


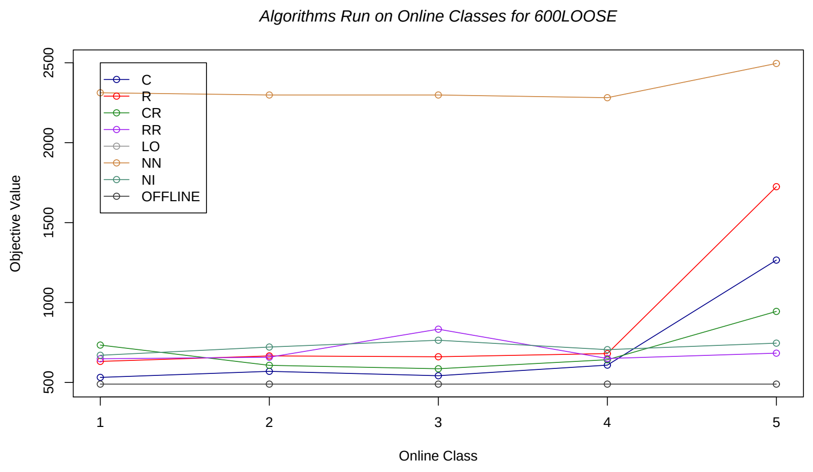
(a) Results for the 180LOOSE instance. The upper figure shows all the algorithms and their solution values. In the lower, the y-axis is limited to a maximum of 6000 to make the results more clear for the algorithms performing well.



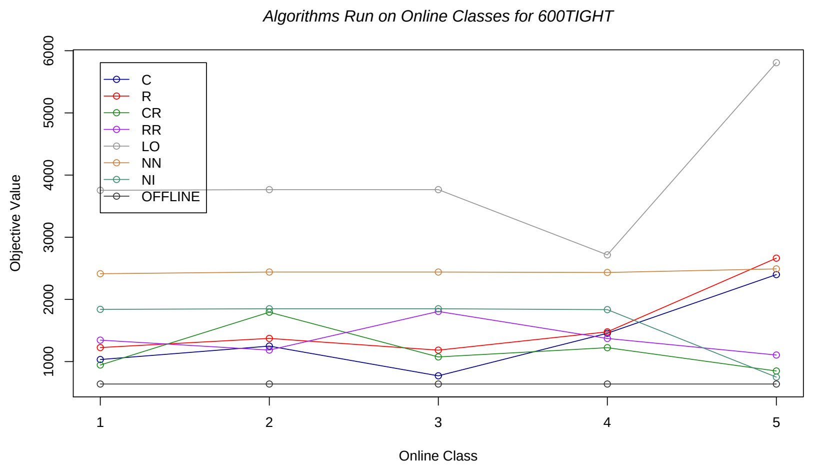
(b) Results for the 180TIGHT instance.

Figure 9.9: Comparison of the Algorithms on the 180 Instances: The figure shows a comparison of the Online Stochastic Algorithms to the oblivious online algorithms and the offline value for the 180 instances.





(a) Results for the 600LOOSE instance. The upper figure shows all the algorithms and their solution values. In the lower, the y-axis is limited to a maximum of 2500 to make the results more clear for the algorithms performing well.



(b) Results for the 600TIGHT instance.

Figure 9.10: Comparison of the Algorithms on the 600 Instances: The figure shows a comparison of the Online Stochastic Algorithms to the oblivious online algorithms and the offline value for the 600 instances.

10 Further Extensions and Perspectives

In Van Hentenryck and Bent [2006], it is mentioned that the results on Online Stochastic Vehicle Routing took them five years to derive. Although this is not comparable to reimplementing their algorithms, the time spent on implementation of the basic framework for the Online Stochastic Algorithms was very extensive, and left much less time for examining and extending the algorithms than could have been desired.

The choice of extension was discretization of the map, as this was thought to be a natural and useful addition to their algorithms. But there were several other features that would be both interesting and useful to implement. This chapter describe these extensions and how they could have been achieved.

10.1 More Extensive Testing of the Algorithms

Generally speaking it would have been desirable with more thorough testing. Due to the long running time of the algorithms, this was not feasible — only a limited amount of testing was possible. Furthermore, since no benchmarks with stochastic information were available, these had to be implemented to challenge different aspects of the algorithms.

Only four different instances were used in the testing of the Online Stochastic Algorithm. This greatly limited how well the algorithms could be tested. Having too much variation in the properties of the instances would make it hard to deduce what properties of the instances affect the algorithms in what way. Having too little variation would not give a fair picture of how well each algorithm perform in general.

The focus was chosen to be on tightness of time window, length of time horizon and number of customers in a route, which is what the constructed instances attempted to challenge. Still, it would have been desirable to test the effect of these properties even more thoroughly, by making similar instances only differing in a single or very few properties. Furthermore, the effect on the algorithm of changes in other properties like positioning of customers, areas with certain properties, demand, etc would have been desirable to examine.

From section 9.6, the need for more instances to base conclusions on became clear. Here the different Online Stochastic Algorithms were compared, but it was not possible to conclude which algorithm performed best, nor under what circumstances one algorithm was superior to others.

The implemented instance generator allows for different areas of the map to have different properties for the customers. This could be used to test the effects of having areas that differed in lateness of requests, amount of demand, size of time windows, etc.. Again, this would require the generation of a series of instances with changes in only a few properties to examine how well the algorithms handle these.

The algorithms of Van Hentenryck and Bent were extended to be able to handle continuous positions on the map by means of discretization. To further examine the effect of discretizing the map, it would have been interesting to compare the algorithms on instances with continuous positioning to the use of discretization on them.

10.2 Improvements in Implementation

The most CPU intense part of the Online Stochastic Algorithms is the generation and, in particular, solving of instances. While it might not be a problem on smaller instances, like the ones solved in this thesis, it could be a problem getting a sufficiently large pool of sampled solutions if the instances are large and very time-demanding to solve. As the Online Stochastic Algorithms base their decisions on this pool, this is a relevant issue to address.

Parallelization: Since the sampled instances can be solved independent of each other, parallelization is an obvious solution to this problem. As an example of how this could be achieved, we have a server and client machines (or CPUs). The clients samples and generate solutions at all times, sending solved instances to the server process. Whenever something happens that the clients need to take into account, like a new request or change to the master plan, the clients are notified, and take this change into account for subsequent sampling and solving of instances. The server process is responsible for finding Consensus, pruning the pool, etc.. The only thing the client processes do is generating plans. Gendreau et al. [1999] have implemented a parallel tabu search for the dynamic VRP, in which a pool of solutions, based on all the visible customers is maintained. The solutions of the pool are continuously optimized concurrently by the clients and consulted by the server process when decisions have to be made. Their work could be used as a starting point when considering a parallel implementation of the Online Stochastic Algorithms.

There can be several gains of generating and solving plans concurrently. First of all, more plans can be generated, since more processing power is available, hence giving the algorithms an improved base for making decisions. When dealing with large instances, it might not be possible to find good solutions within the timespan of a single time step. When distributing the job of solving the sampled instances to other processors, these can be allowed to span over multiple time units, as long as no new customers arrive or addition of new customers to the master plan occurs. This means that the offline algorithm has time to find good solutions even on large, time consuming instances.

Solving Across Time steps: Allowing the solving of plans to run across time steps is possible without parallelization. For this, the optimization would be allowed to run uninterrupted until either a new request is made or a vehicle is idle. The next idle time of a vehicle is known with certainty every time a customer is assigned to it. In terms of new request there is no way of foreseeing this exactly, so some type of interrupt function would have to be implemented to notify the algorithm that it has no more time to solve instances. Furthermore, the algorithm would have to keep track of the current time, t, during optimization, so no illegal moves are made in the optimization.

10.3 Extensions to the Algorithms

Solving Large Instances: The instances solved in this thesis only contains 50 customers. In a real life application instances would often be many times bigger. Since the VRP is in the class of NP, a big increase in size would lead to a dramatic increase in running time. If the size of the instances to be solved are too big to find reasonably good solutions in the available timespan, a solution would be to only sample a part of the time horizon. So if the current time is t, one could sample until time $t + x$ and only consider the customers that need service in the next x time units. This obviously decreases the instance size and (depending on the size of x) allows the algorithms to find good solutions within the allowed time. Since only a part of the time horizon is considered, in the end, the solutions might not be as good as if the entire time horizon was considered, but it allows to solve much bigger instances (many customers). This principle can also be applied to handle problems in which there is no finite time horizon, ie. the instance is continuous.

Switching Offline Algorithm: Throughout the execution of the algorithm, increasing parts of the solution will be fixated (namely the part preceding time t). The remaining part, that has to be solved, obviously becomes smaller and consequently faster to solve. Optimal solvers have been considered infeasible as a sub-procedure to the Online Stochastic Algorithms due to their long running times compared to heuristics. But at some point in the execution of the Online Stochastic Algorithm, the remaining part of the instance to solve will be sufficiently small, to let optimal solvers be used within the timespan allowed. Having the offline solver switch to a branch and bound algorithm at some point in time, for example, would start giving optimally solved sampled instances, which could have a positive effect on the solution quality of the master plan.

Maintaining Large Pool size: A potential problem for the Online Stochastic Algorithms described in this thesis is their dependency on the pool of sampled plans for making decisions. When this is very small, the decisions made by the algorithm will be worse. To avoid small pools one could attempt to generate more plans, by means of sampling only parts of the time horizon, making the instance size smaller, as described above or use a faster algorithm finding solutions of lower quality.

Another strategy would be to attempt to minimize the number of plans pruned. Instead of pruning all the plans that do not agree with the change in the master plan, the algorithm could attempt to fit the sampled plans to these changes. As a simple example of this, if the sampled plan γ_s does not agree on the consensus of a route, the customer of γ_s could be attempted replaced by the consensus customer, hence avoiding to prune γ_s. On of course has to ensure that the consensus customer is not present elsewhere in γ_s and that the replacement is feasible in terms of timewindows, etc.

This, and more elaborate schemes to fit the sampled plans to the choice of consensus might help the algorithm keep a bigger pool of sampled plans, and hence make better decisions when adding customers to the master plan.

10.4 Extensions to the Model

In this chapter, mostly algorithmic and implementational improvements have been considered but there are some extensions to the model that also would be interesting to examine. These will be discussed here.

Moving from completely offline problems to the online stochastic VRP considered in this thesis, is a big step in terms of being able to model real life problems more realistically. However, as previously mentioned, it might not be realistic to know the service time, travel times, demand, etc. as soon as the request is made. There might be delays in traffic due to heavy traffic, accidents, or the like. In terms of service time, it might not be possible to predict, before arriving at the customer. The same applies to demand. Including these uncertainties in the model would be another big step towards making it more realistic and applicable in practical applications. Due to the nature of the Online Stochastic Algorithms, this extension would be relatively easy to implement. Since the master plan is only extended when a vehicle is idle, the extension of a visit (length of service time) for example, would only shift the time at which the vehicle is idle and a new decision has to be made. There is not a plan extending the visit, that could become infeasible due to a prolonging of a visit (except of course if the visit extends as far as leaving too little time to return to the depot within the time horizon h). When a change in eg. travel time occurs this is saved in the master plan and the pool of sampled plans is then pruned for those that cannot conform to these new changes. This would of course mean that the pool of sampled plans would be pruned more often, leaving a smaller pool to base decisions on, which in the end has a negative effect on the solution quality of the algorithm. On the other hand, the algorithm would be able to handle much more realistic problems. Besides this, the algorithm already has the functionality for a qualified guess on the properties of a customer (sampling), and for coping with changes in the properties of a customer (by pruning). This would therefore be a interesting extension to the model, that could be handled in a very natural way by the Online Stochastic Algorithms.

11 Conclusion

This thesis is based on Van Hentenryck and Bents work on online stochastic vehicle routing [Van Hentenryck and Bent, 2006]. It focuses on taking advantage of stochastic information when solving the dynamic VRP. The algorithms described by Van Hentenryck and Bent introduce a new approach, in which the unknown parts of the instances are sampled using the stochastic knowledge available. This has two advantages: it allows regular offline algorithms to be used to solve the partly sampled instances, and secondly, if several of these partly sampled instances are solved, this can be used as a basis for guiding the vehicles throughout the execution.

For this thesis, a different offline algorithm than that of Van Hentenryck and Bent is used, namely the Attribute Based Hill Climber (ABHC). Furthermore, the Online Stochastic Algorithms are extended to use discretization of the map, allowing the algorithms to handle continuous positioning of customers. The results of the Online Stochastic Algorithms using discretization and the ABHC are compared to the solutions found by three oblivious online algorithms as well as the offline solution to the instances. Since no instances with known stochastic information could be found, these had to be generated. This is done by the use of an instance template, implemented for this thesis.

The discretization seems to have a positive effect on the solutions found. Since only four instances were used for testing, a more thorough evaluation of the effect of discretization would be desirable. In particular, a comparison of the solution found by the algorithms with and without the use of discretization on instances with a continuous positioning of the customers would be interesting.

Although the Online Stochastic Algorithms in general perform well, the results are somewhat disappointing compared to the impression one had from the book of Van Hentenryck and Bent [2006], in particular because the discretization actually seems to improve the algorithms described in the book. There can be several reason for the difference in results:

- The implementation of the algorithm for this thesis could be less efficient than that of Van Hentenryck and Bent.
- Although both the book and the articles of Van Hentenryck and Bent were consulted, certain parts of the algorithms might have been misunderstood or misimplemented, which could lead to worse results.
- The instances the algorithms tested on differs from this thesis to Van Hentenryck and Bent. This includes both structure of the instances, and the precision of sampling. Unfortunately it has not been possible to reconstruct their testbase, based on the information available in the book and articles.
- In Van Hentenryck and Bent [2006], the Online Stochastic Algorithms were only compared to NN and an algorithm very similar to LO. These find significantly worse solutions

than NI used in this thesis, which this explains the difference of results of the Online Stochastic Algorithms over the oblivious online algorithms. Still, this does not fully explain the difference in results, since the internal ranking of the Online Stochastic Algorithms according to the solution quality they find is also different.

The question remains whether it is desirable to use Online Stochastic Algorithms for the solving an online VRP. As described, more thorough testing of the algorithms would be required to answer this question properly, but based on this thesis, the answer is that in most cases it is. In the end, it depends on how important it is to find good solutions, and how much time one is willing to spend on implementing the algorithms. An implementation of the framework of the online VRP as well as the NI algorithm would take around a week or so. For implementing an Online Stochastic Algorithm with pruning, sampling of customers, consensus calculation etc. neeeded for it, would take significantly longer. Furthermore, the precision of the available stochastic data has a big influence on whether the Online Stochastic Algorithms is to prefer over oblivious online algorithms.

The ideas described in Chapter 10: “Further Extensions and Perspectives” might help make the algorithm more efficient and furthermore extend their use to model and solve more realistic applications, and hence make them more suitable than a simple oblivious online algorithm. These are natural extension to the model and algorithms, that would be very interesting to examine further.

Bibliography

P. Van Hentenryck and R. Bent. *Online Stochastic Combinatorial Optimization*. The MIT Press, 1 edition, 2006.

M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger. Subset-row inequalities applied to the vehicle routing problem with time windows. *Operations Research*, 56(2):497–511, 2008.

M.M Solomon. Solomon benchmarks: Best known solutions identified by heuristics (http://web.cba.neu.edu/˜msolomon/heuristi.htm), March 2005.

J. Larsen. *Parallelization of the vehicle routing problem with time windows.* Institute of Mathematical Modelling, Technical University of Denmark, 1999.

B. De Backer, V. Furnon, P. Kilby, P. Prosser, and P. Shaw. Local search in constraint programming: Application to the Vehicle Routing Problem. In *Constraint Programming 97, Proceedings of the Workshop on Industrial Constraint-Directed Scheduling*, 1997.

R.W. Bent and P. Van Hentenryck. Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers. *Operations Research*, 52(6):977, 2004a.

R. Bent and P. Van Hentenryck. Dynamic Vehicle Routing with Stochastic requests. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 1362–1363. Lawrence Erlbaum Associates Ltd, 2003.

R. Bent and P. Van Hentenryck. The Value of Consensus in Online Stochastic Scheduling. *ICAPS 2004*, 2004b.

R. Bent and P. Van Hentenryck. Regrets Only! Online Stochastic Optimization under Time Constraints. In *Proceedings of the National Conference on Artificial Intelligence*, pages 501–506. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2004c.

R. Bent and P. Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530, 2004d.

R. Bent and P. Van Hentenryck. Online stochastic and robust optimization. In *Proceeding of the 9th Asian Computing Science Conference (ASIAN'04)*, 2004e.

R. Bent, I. Katriel, and P. Van Hentenryck. Sub-optimality Approximations. *Lecture Notes in Computer Science*, 3709:122, 2005.

R. Bent and P. Van Hentenryck. Online Stochastic Optimization without Distributions. In *Proceedings of the 15th International Conference on Automated Planning & Scheduling (ICAPS)*, 2005.

R. Bent and P.V. Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers and Operations Research*, 33(4):875–893, 2006.

H. Chang, R. Givan, and E. Chong. On-line scheduling via sampling. *Artificial Intelligence Planning and Scheduling (AIPS)(Breckenridge, Colorado, 2000)*, pages 62–71.

P. Toth and D. Vigo. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, PA, USA, 2002.

Leiserson C.E. Rivest R.L. Cormen, T.H. and C. Stein. *Introduction to algorithms*. MIT press, 2001.

MM Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

H.N. Psaraftis. Dynamic vehicle routing problems. *Vehicle Routing: Methods and Studies*, 16: 223–248, 1988.

W.R. Stewart. The Delivery Truck Routing Problem with Stochastic Demands. In *ORSA/TIMS meeting*, November 1976.

T.M. Cook and R.A. Russell. A Simulation and Statistical Analysis of Stochastic Vehicle Routing with Timing Constraints. *Decision Sciences*, 9(4):673–687, 1978.

L.M. Hvattum, A. Lokketangen, and G. Laporte. A branch-and-regret heuristic for stochastic and dynamic vehicle routing problems. *Network-New York-*, 49(4):330, 2007.

G. Clarke and JW Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

O. Braysy and M. Gendreau. Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms. *Transportation Science*, 39(1):104–118, 2005a.

O. Braysy and M. Gendreau. Vehicle Routing Problem with Time Windows, Part II: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005b.

G. Ioannou, M. Kritikos, and G. Prastacos. A greedy look-ahead heuristic for the vehicle routing problem with time windows. *The Journal of the Operational Research Society*, 52(5):523–537, 2001.

J.B. Atkinson. A greedy look-ahead heuristic for combinatorial optimization: an application to vehicle scheduling with time windows. *Journal-Operational Research Society*, 45:673–673, 1994.

I.M. Whittley and G.D. Smith. The attribute based hill climber. *Journal of Mathematical Modelling and Algorithms*, 3(2):167–178, 2004.

U. Derigs and R. Kaiser. Applying the attribute based hill climber heuristic to the vehicle routing problem. *European Journal of Operational Research*, 177(2):719–732, 2007.

Jesper Nikolajsen. Algorithms for Optimizing Work Schedules for Service Personnel in the Home Care Sector. Master’s thesis, Department of Mathermatics and Computer Science (IMADA), University of Southern Denmark, 2009.

H.H. Hoos and T. Stützle. *Stochastic local search: Foundations and applications*. Morgan Kaufmann, 2005.

A. Larsen, O. Madsen, and M. Solomon. Partially dynamic vehicle routing - models and algorithms. *Journal of the Operational Research Society*, 53:637–646, jun 2002.

M. Gendreau, F. Guertin, J.Y. Potvin, and E. Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, 33(4):381, 1999.

A Detailed Test Output

A.1 ILS Tuning: Graphs

A.1.1 Tuning Permutation by Neighbourhood

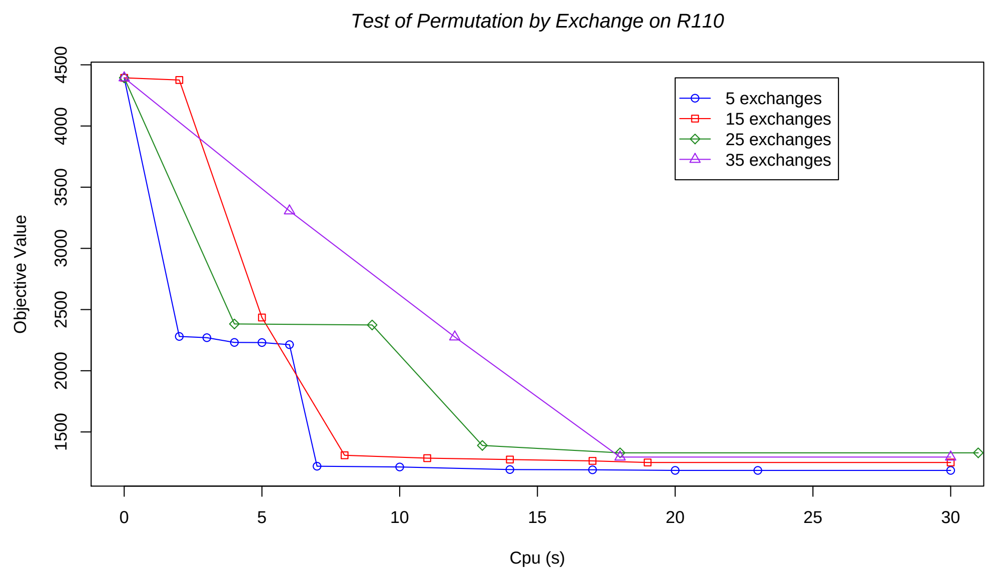


Figure A.1: ILS: Permutation by Neighbourhood: Large graph of tuning on R110 from section 6.4.2

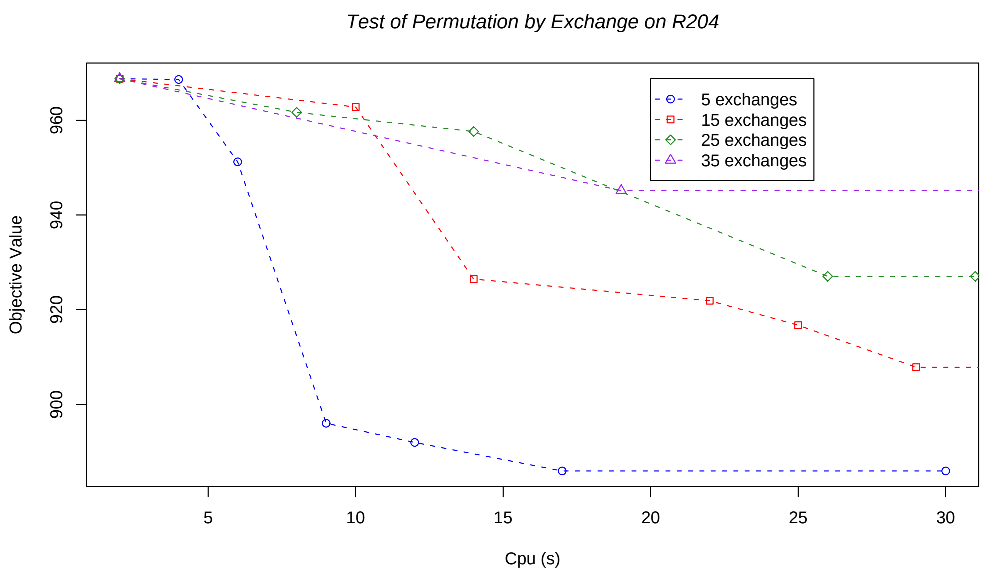


Figure A.2: ILS: Permutation by Neighbourhood: Large graph of tuning on R204 from section 6.4.2

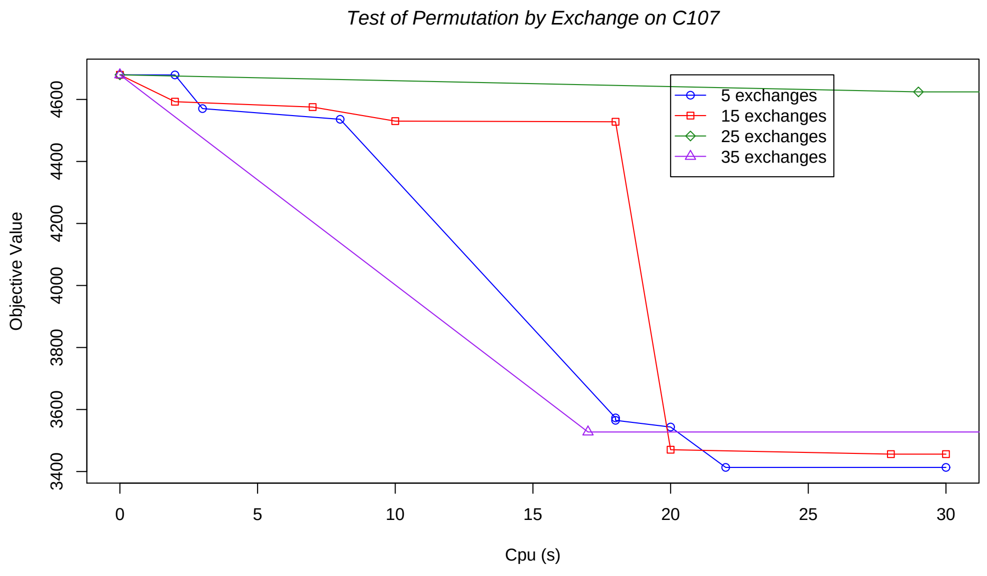


Figure A.3: ILS: Permutation by Neighbourhood: Large graph of tuning on C107 from section 6.4.2

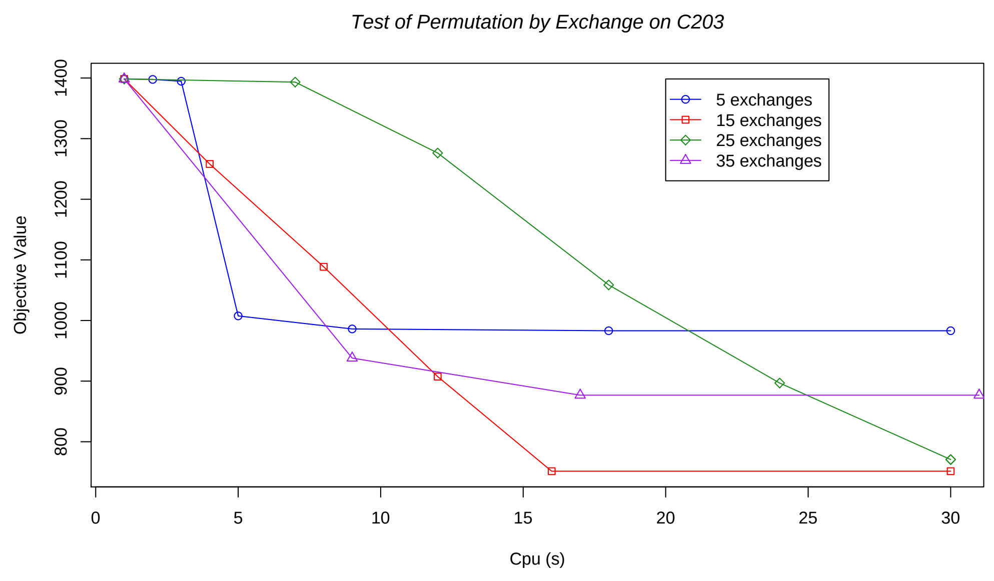


Figure A.4: ILS: Permutation by Neighbourhood: Large graph of tuning on C203 from section 6.4.2

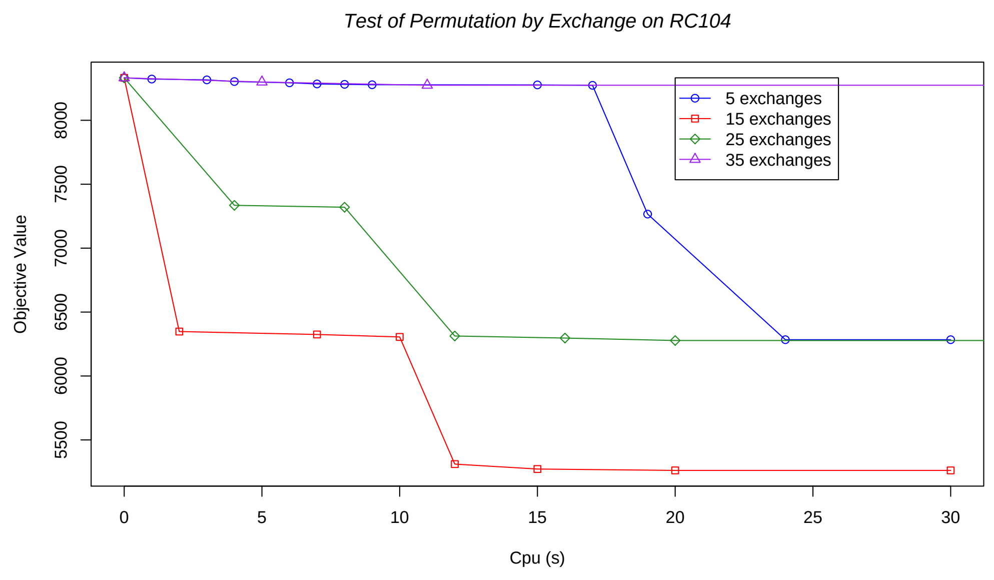


Figure A.5: ILS: Permutation by Neighbourhood: Large graph of tuning on RC104 from section 6.4.2

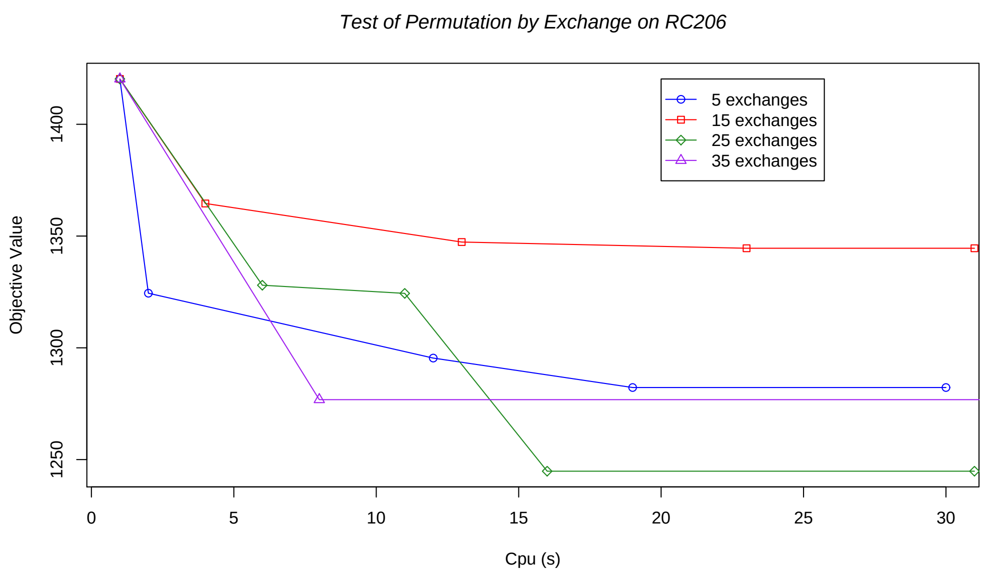


Figure A.6: ILS: Permutation by Neighbourhood: Large graph of tuning on RC206 from section 6.4.2

A.1.2 Tuning Permutation by Removal

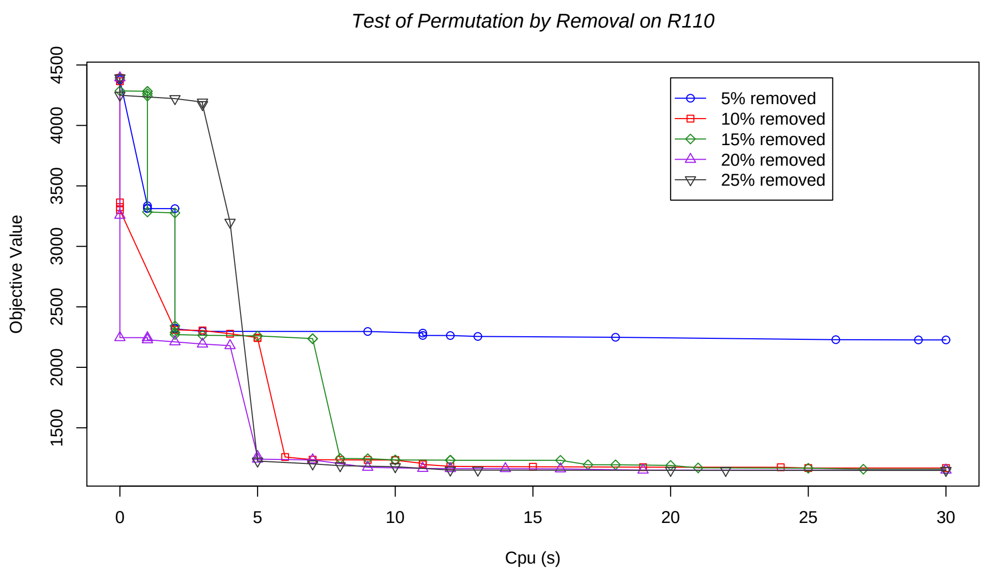


Figure A.7: ILS: Permutation by Removal: Large graph of tuning on R110 from section 6.4.2

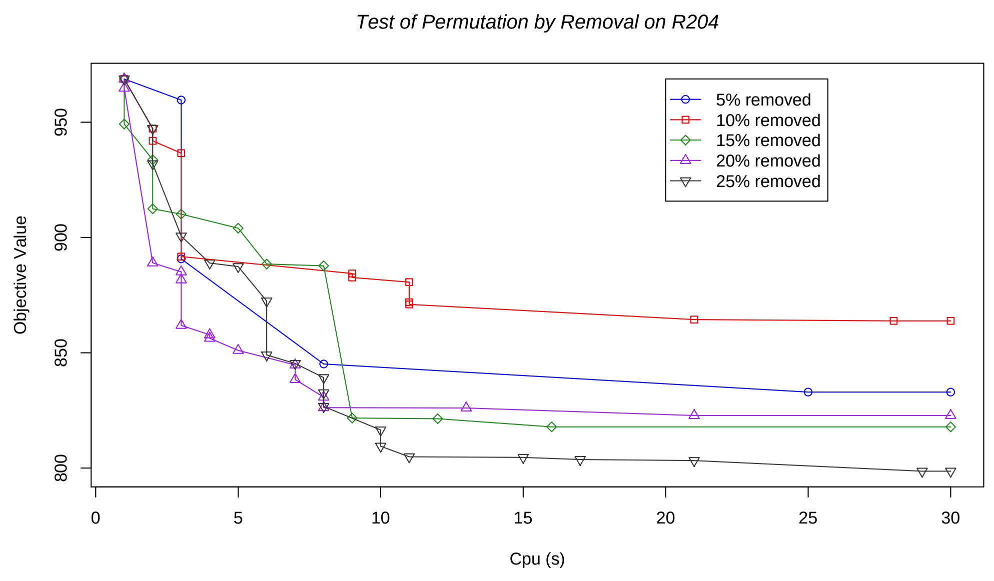


Figure A.8: ILS: Permutation by Removal: Large graph of tuning on R204 from section 6.4.2

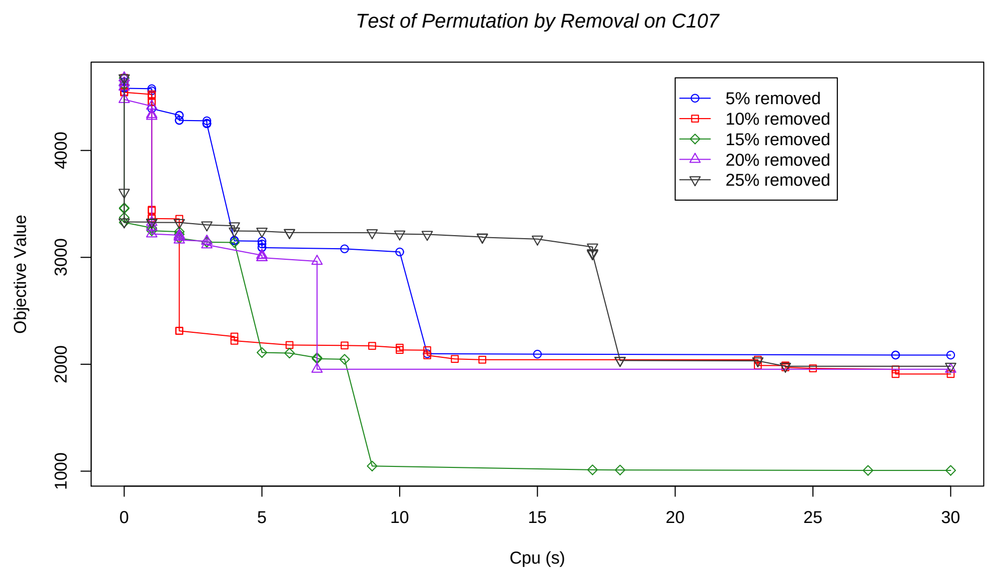


Figure A.9: ILS: Permutation by Removal: Large graph of tuning on C107 from section 6.4.2

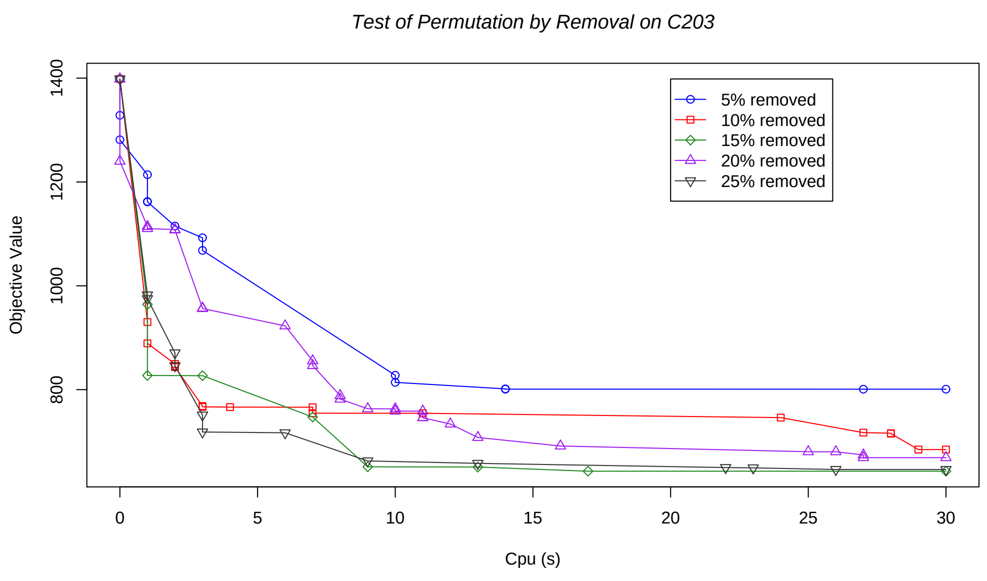


Figure A.10: ILS: Permutation by Removal: Large graph of tuning on C203 from section 6.4.2

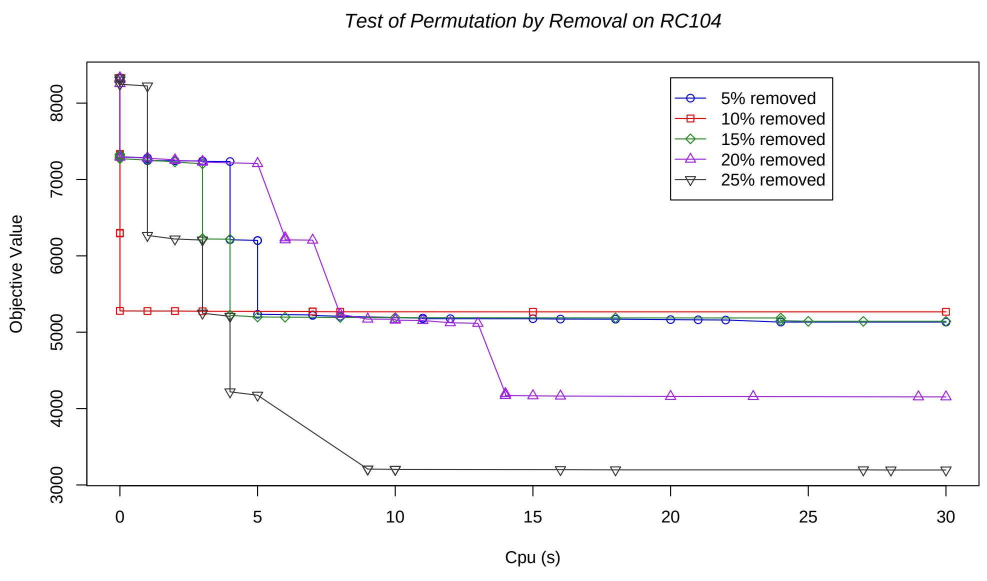


Figure A.11: ILS: Permutation by Removal: Large graph of tuning on RC104 from section 6.4.2

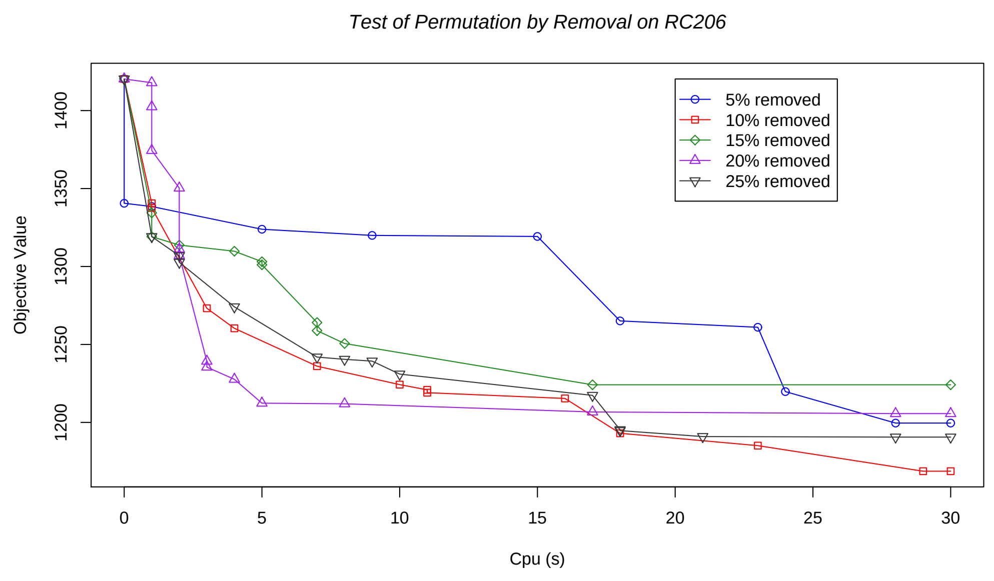


Figure A.12: ILS: Permutation by Removal: Large graph of tuning on RC206 from section 6.4.2

A.1.3 Tuning Permutation by Removal and Neighbourhood

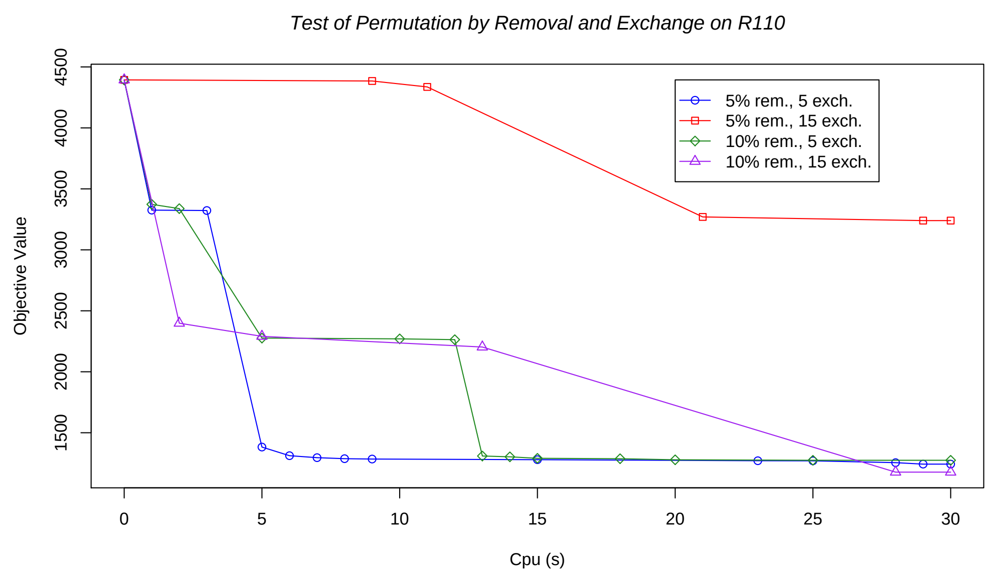


Figure A.13: ILS: Permutation by Removal and Permutation: Large graph of tuning on R110 from section 6.4.2

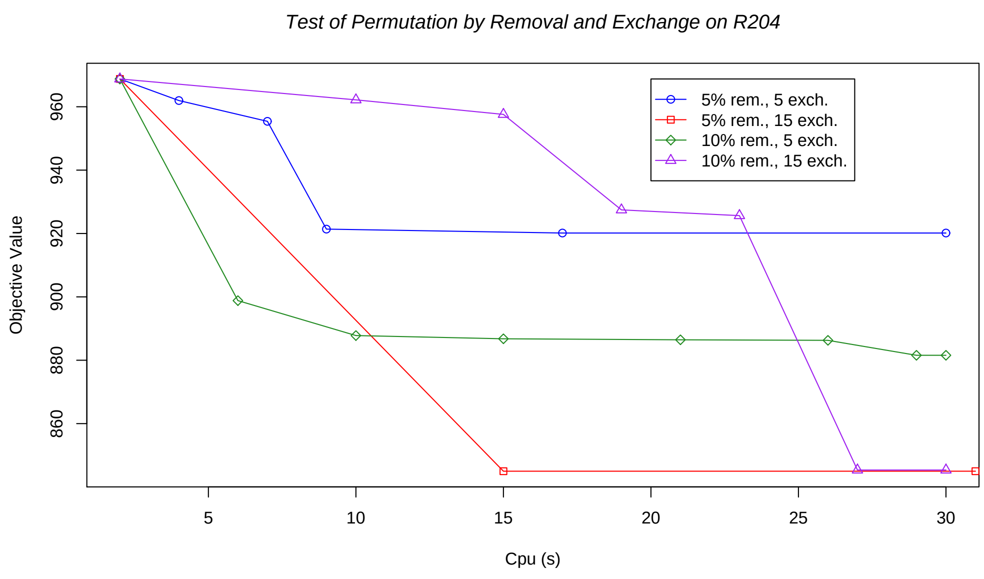


Figure A.14: ILS: Permutation by Removal and Permutation: Large graph of tuning on R204 from section 6.4.2

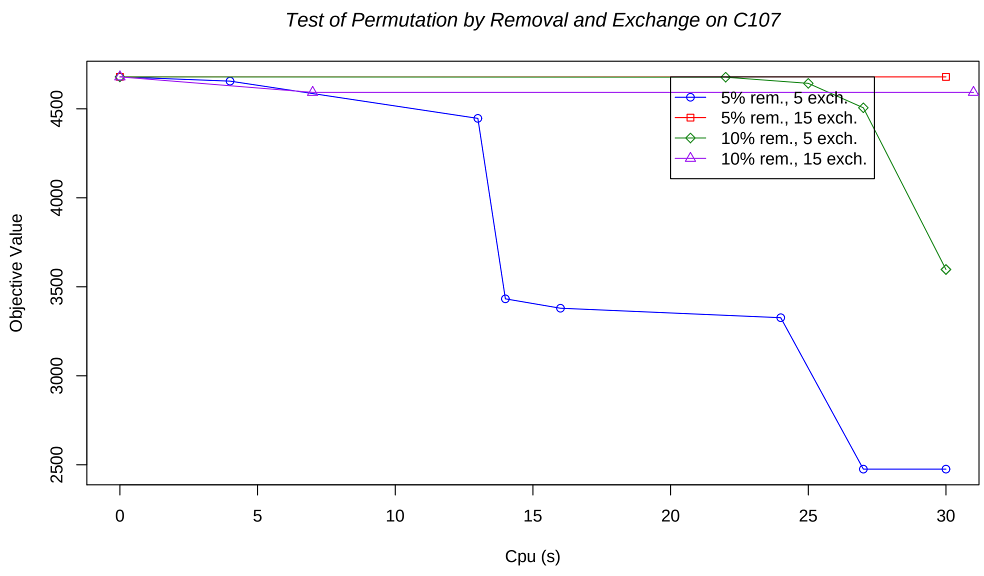


Figure A.15: ILS: Permutation by Removal and Permutation: Large graph of tuning on C107 from section 6.4.2

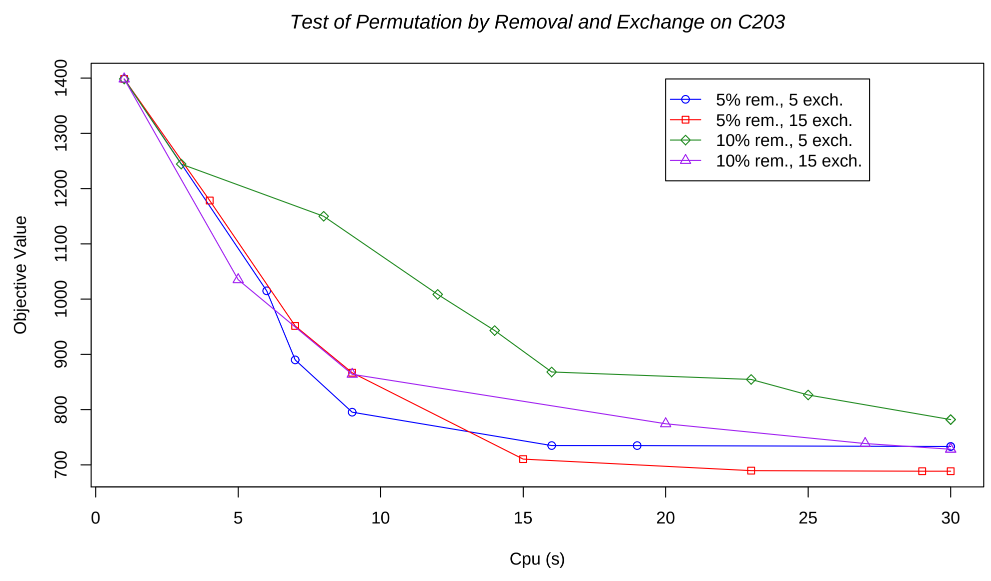


Figure A.16: ILS: Permutation by Removal and Permutation: Large graph of tuning on C203 from section 6.4.2

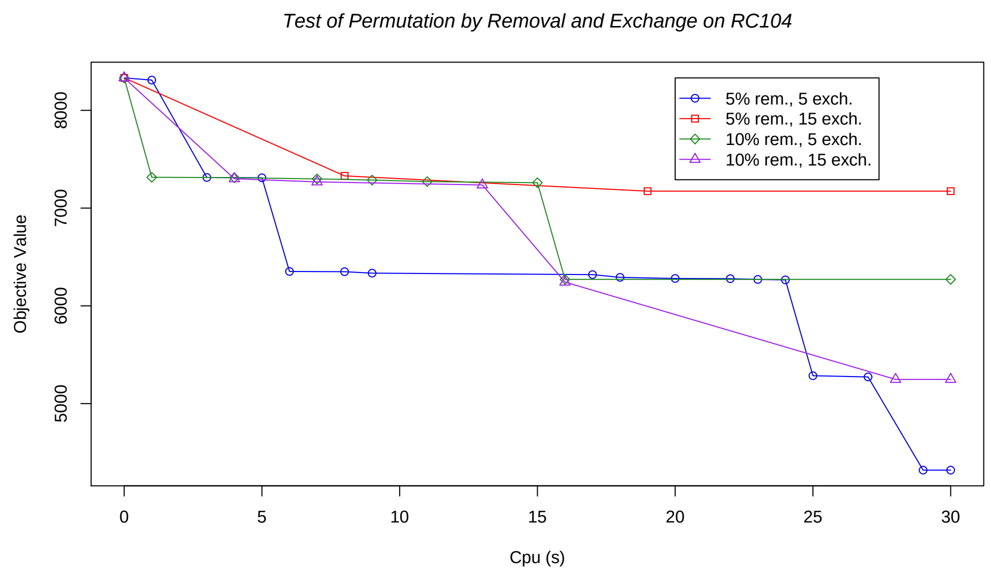


Figure A.17: ILS: Permutation by Removal and Permutation: Large graph of tuning on RC104 from section 6.4.2

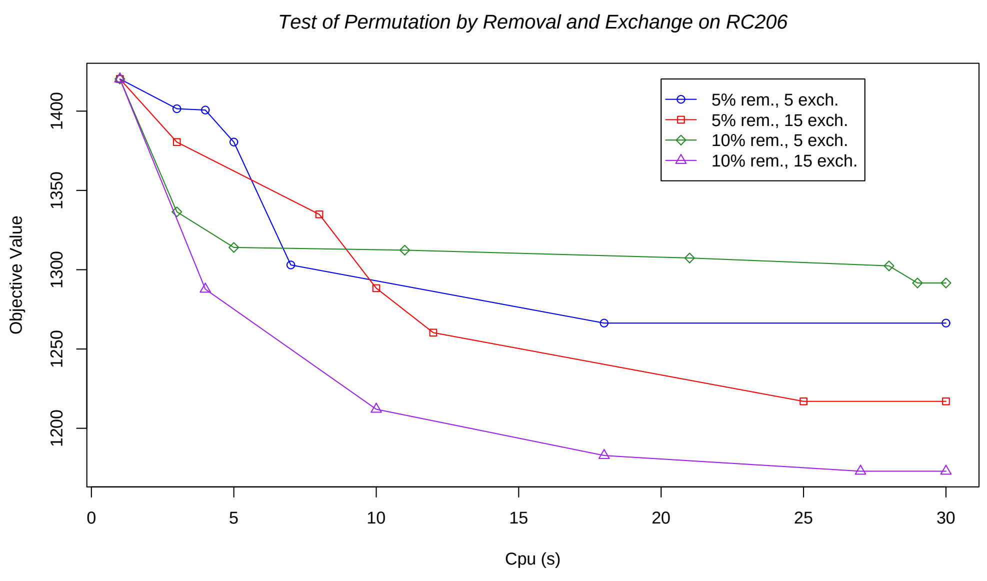


Figure A.18: ILS: Permutation by Removal and Permutation: Large graph of tuning on RC206 from section 6.4.2

A.2 Detailed Results of Regret Tuning

Tuning of Regret with Discretization Size 35

```
RR_PLAN.dat:
------------------
Unserved Length Obj Instance InstanceNr
3 1102.62 3001102622.00 180Loose 1
1 1133.04 1001133042.00 180Tight 2
0 673.39 673390.00 600Loose 3
2 842.71 2000842714.00 600Tight 4


RR_PLAN_SAMPLED.dat:
------------------
Unserved Length Obj Instance InstanceNr
2 1028.47 2001028474.00 180Loose 1
2 1176.95 2001176950.00 180Tight 2
1 725.55 1000725546.00 600Loose 3
1 815.24 1000815238.00 600Tight 4

RR_ROUTE.dat:
------------------
Unserved Length Obj Instance InstanceNr
1 1105.05 1001105051.00 180Loose 1
2 1117.85 2001117849.00 180Tight 2
0 670.13 670129.00 600Loose 3
0 834.14 834139.00 600Tight 4

RR_ROUTE_SAMPLED.dat:
------------------
```

```
Unserved Length Obj Instance InstanceNr
1 1152.21 1001152212.00 180Loose 1
0 1131.31 1131314.00 180Tight 2
0 711.60 711596.00 600Loose 3
0 882.55 882554.00 600Tight 4

R_PLAN.dat:
------------------
Unserved Length Obj Instance InstanceNr
1 891.47 1000891471.00 180Loose 1
2 1040.74 2001040739.00 180Tight 2
0 655.92 655924.00 600Loose 3
3 721.26 3000721264.00 600Tight 4

R_ROUTE.dat:
------------------
Unserved Length Obj Instance InstanceNr
1 964.30 1000964295.00 180Loose 1
0 1019.60 1019600.00 180Tight 2
0 606.01 606009.00 600Loose 3
2 764.27 2000764268.00 600Tight 4
```

Tuning of Regret with Discretization Size 24

```
RR_ROUTE_SAMPLED.dat:
------------------
Unserved Length Obj Instance InstanceNr
0 1046.34 1046343.00 180Loose 1
0 1228.84 1228838.00 180Tight 2
0 705.56 705562.00 600Loose 3
2 865.94 2000865939.00 600Tight 4

RR_ROUTE.dat:
-------------------
Unserved Length Obj Instance InstanceNr
2 1049.08 2001049080.00 180Loose 1
1 1095.66 1001095657.00 180Tight 2
0 731.38 731378.00 600Loose 3
0 777.68 777683.00 600Tight 4

RR_PLAN_SAMPLED.dat:
-------------------
Unserved Length Obj Instance InstanceNr
2 1098.15 2001098152.00 180Loose 1
1 1127.31 1001127306.00 180Tight 2
1 666.78 1000666782.00 600Loose 3
0 854.07 854069.00 600Tight 4

RR_PLAN.dat:
-------------------
Unserved Length Obj Instance InstanceNr
0 1154.21 1154205.00 180Loose 1
1 1195.04 1001195041.00 180Tight 2
0 679.78 679780.00 600Loose 3
1 803.02 1000803019.00 600Tight 4

R_ROUTE.dat:
------------------
Unserved Length Obj Instance InstanceNr
1 1007.54 1001007535.00 180Loose 1
```

```
0 1018.47 1018471.00 180Tight 2
0 632.26 632256.00 600Loose 3
1 801.81 1000801805.00 600Tight 4

R_PLAN.dat:
------------------
Unserved Length Obj Instance InstanceNr
1 947.48 1000947484.00 180Loose 1
0 1067.22 1067216.00 180Tight 2
0 694.49 694489.00 600Loose 3
2 877.87 2000877865.00 600Tight 4
```

A.3 Detailed Results of Discretization Tuning

Tuning of Discretization for CR

```
RR_70.dat:
------------------
Unserved Length Obj Instance InstanceNr
1 1067.10 1001067102.00 180Loose 1
2 1095.24 2001095235.00 180Tight 2
0 613.62 613621.00 600Loose 3
0 899.32 899320.00 600Tight 4

RR_35.dat:
------------------
Unserved Length Obj Instance InstanceNr
0 1078.77 1078769.00 180Loose 1
1 1276.54 1001276542.00 180Tight 2
0 636.52 636523.00 600Loose 3
0 920.48 920483.00 600Tight 4

RR_24.dat:
------------------
Unserved Length Obj Instance InstanceNr
1 1004.18 1001004176.00 180Loose 1
0 1195.10 1195099.00 180Tight 2
0 669.68 669677.00 600Loose 3
1 755.92 1000755915.00 600Tight 4

RR_18.dat:
------------------
Unserved Length Obj Instance InstanceNr
0 959.49 959492.00 180Loose 1
0 1080.98 1080976.00 180Tight 2
1 719.17 1000719166.00 600Loose 3
1 833.90 1000833897.00 600Tight 4

RR_10.dat:
------------------
Unserved Length Obj Instance InstanceNr
1 979.27 1000979271.00 180Loose 1
1 1153.89 1001153894.00 180Tight 2
0 630.90 630902.00 600Loose 3
0 839.03 839034.00 600Tight 4
```

Tuning of Discretization for RR

```
RR_70.dat:
------------------
Unserved Length Obj Instance InstanceNr
```

```
1 1063.16 1001063155.00 180Loose 1
1 1246.51 1001246512.00 180Tight 2
0 703.13 703134.00 600Loose 3
0 864.07 864072.00 600Tight 4

RR_35.dat:
------------------
Unserved Length Obj Instance InstanceNr
1 1080.83 1001080830.00 180Loose 1
2 1130.79 2001130794.00 180Tight 2
0 708.03 708030.00 600Loose 3
2 812.51 2000812513.00 600Tight 4


RR_24.dat:
------------------
Unserved Length Obj Instance InstanceNr
1 1036.54 1001036541.00 180Loose 1
1 1119.06 1001119061.00 180Tight 2
0 684.46 684455.00 600Loose 3
0 870.27 870269.00 600Tight 4


RR_18.dat:
------------------
Unserved Length Obj Instance InstanceNr
1 1173.41 1001173412.00 180Loose 1
1 1119.30 1001119298.00 180Tight 2
0 713.80 713796.00 600Loose 3
3 746.89 3000746890.00 600Tight 4


RR_10.dat:
------------------
Unserved Length Obj Instance InstanceNr
2 1049.83 2001049830.00 180Loose 1
1 1158.47 1001158468.00 180Tight 2
0 686.78 686781.00 600Loose 3
1 797.24 1000797236.00 600Tight 4
```

A.4 Detailed Results from Historical Sampling Tests

A.4.1 180LOOSE

```
Algorithm SamplingType Unassigned Length Obj Instance InstanceNr
C 0 3 934.74 3934739.00 180Loose 1
C 1 2 960.43 2960432.00 180Loose 1
C 2 1 982.47 1982470.00 180Loose 1
C 3 1 990.97 1990967.00 180Loose 1
C 4 1 900.91 1900907.00 180Loose 1
R 0 3 954.93 3954933.00 180Loose 1
R 1 2 1021.78 3021776.00 180Loose 1
R 2 4 927.51 4927512.00 180Loose 1
R 3 1 947.14 1947138.00 180Loose 1
R 4 0 913.42 913419.00 180Loose 1
CR 0 4 1050.41 5050412.00 180Loose 1
CR 1 2 1049.40 3049396.00 180Loose 1
CR 2 3 1064.66 4064664.00 180Loose 1
CR 3 2 1017.42 3017422.00 180Loose 1
CR 4 2 962.86 2962859.00 180Loose 1
```

```
RR 0 3 1044.91 4044906.00 180Loose 1
RR 1 2 1060.95 3060952.00 180Loose 1
RR 2 3 1099.25 4099246.00 180Loose 1
RR 3 5 992.82 5992816.00 180Loose 1
RR 4 1 938.64 1938640.00 180Loose 1
```

A.4.2 180TIGHT

```
Algorithm SamplingType Unassigned Length Obj Instance InstanceNr
C 0 4 992.83 4992834.00 180Tight 2
C 1 2 1011.54 3011540.00 180Tight 2
C 2 2 967.90 2967897.00 180Tight 2
C 3 2 1021.06 3021060.00 180Tight 2
C 4 1 999.06 1999060.00 180Tight 2
R 0 3 1033.51 4033514.00 180Tight 2
R 1 4 920.81 4920814.00 180Tight 2
R 2 3 954.13 3954128.00 180Tight 2
R 3 3 964.84 3964842.00 180Tight 2
R 4 0 1007.16 1007163.00 180Tight 2
CR 0 6 1172.93 7172925.00 180Tight 2
CR 1 3 1116.77 4116773.00 180Tight 2
CR 2 1 1095.22 2095222.00 180Tight 2
CR 3 6 1150.88 7150883.00 180Tight 2
CR 4 1 1014.15 2014154.00 180Tight 2
RR 0 9 1102.96 10102956.00 180Tight 2
RR 1 3 1049.67 4049670.00 180Tight 2
RR 2 3 1102.86 4102862.00 180Tight 2
RR 3 2 1172.25 3172248.00 180Tight 2
RR 4 0 1033.74 1033736.00 180Tight 2
```

A.4.3 600LOOSE

```
Algorithm SamplingType Unassigned Length Obj Instance InstanceNr
C 0 1 594.72 1594721.00 600Loose 3
C 1 0 595.98 595981.00 600Loose 3
C 2 1 587.09 1587090.00 600Loose 3
C 3 0 620.63 620631.00 600Loose 3
C 4 0 599.86 599855.00 600Loose 3
R 0 0 690.46 690462.00 600Loose 3
R 1 0 680.88 680879.00 600Loose 3
R 2 0 743.67 743665.00 600Loose 3
R 3 0 782.37 782367.00 600Loose 3
R 4 0 683.49 683493.00 600Loose 3
CR 0 2 835.24 2835239.00 600Loose 3
CR 1 0 863.53 863532.00 600Loose 3
CR 2 0 970.31 970310.00 600Loose 3
CR 3 0 789.38 789384.00 600Loose 3
CR 4 0 555.04 555041.00 600Loose 3
RR 0 4 815.31 4815308.00 600Loose 3
RR 1 0 1039.61 1039614.00 600Loose 3
RR 2 0 1008.56 1008563.00 600Loose 3
RR 3 0 872.95 872953.00 600Loose 3
RR 4 0 657.28 657278.00 600Loose 3
```

A.4.4 600TIGHT

```
Algorithm SamplingType Unassigned Length Obj Instance InstanceNr
C 0 1 819.78 1819776.00 600Tight 4
C 1 1 758.66 1758663.00 600Tight 4
C 2 0 834.93 834930.00 600Tight 4
C 3 1 774.48 1774477.00 600Tight 4
C 4 0 730.74 730735.00 600Tight 4
```

```
R 0 1 853.46 1853455.00 600Tight 4
R 1 2 790.55 2790545.00 600Tight 4
R 2 2 777.47 2777468.00 600Tight 4
R 3 1 752.36 1752356.00 600Tight 4
R 4 0 783.29 783291.00 600Tight 4
CR 0 2 1263.94 3263943.00 600Tight 4
CR 1 0 1172.01 1172007.00 600Tight 4
CR 2 0 1325.67 1325668.00 600Tight 4
CR 3 0 1251.81 1251808.00 600Tight 4
CR 4 1 767.88 1767884.00 600Tight 4
RR 0 1 1147.45 2147447.00 600Tight 4
RR 1 2 1246.77 3246766.00 600Tight 4
RR 2 1 1186.26 2186255.00 600Tight 4
RR 3 0 1249.93 1249930.00 600Tight 4
RR 4 1 799.70 1799703.00 600Tight 4
```

A.5 Detailed Results of the Comparison of Algorithms

A.5.1 180LOOSE

Offline solution

```
Algorithm OnlineClass Unassigned Length Obj Instance InstanceNr
OFF 0 0 787468.0 787468.0 180Loose 1
```

Oblivious Online Solutions

```
Algorithm OnlineClass Unassigned Length Obj Instance InstanceNr runNr
RR 1 0 974.11 974113.00 180Loose 1 1
RR 1 0 960.16 960156.00 180Loose 1 2
RR 1 1 947.62 1947620.00 180Loose 1 3
RR 1 2 922.93 2922929.00 180Loose 1 5
RR 1 2 868.17 2868173.00 180Loose 1 4
RR 1 1 978.29 1978285.00 180Loose 1 6
RR 2 2 946.71 2946705.00 180Loose 1 2
RR 2 2 889.21 2889209.00 180Loose 1 1
RR 2 3 915.87 3915871.00 180Loose 1 3
RR 2 2 897.23 2897231.00 180Loose 1 5
RR 2 1 917.26 1917264.00 180Loose 1 6
RR 2 2 1005.61 3005607.00 180Loose 1 4
RR 3 2 950.71 2950709.00 180Loose 1 2
RR 3 1 917.45 1917454.00 180Loose 1 1
RR 3 1 922.93 1922928.00 180Loose 1 3
RR 3 2 938.22 2938221.00 180Loose 1 5
RR 3 1 904.94 1904939.00 180Loose 1 6
RR 3 1 884.84 1884836.00 180Loose 1 4
RR 4 1 986.14 1986140.00 180Loose 1 2
RR 4 2 991.30 2991298.00 180Loose 1 1
RR 4 4 882.49 4882490.00 180Loose 1 3
RR 4 2 862.86 2862863.00 180Loose 1 6
RR 4 2 887.68 2887675.00 180Loose 1 5
RR 4 1 933.71 1933705.00 180Loose 1 4
RR 5 2 923.31 2923308.00 180Loose 1 2
RR 5 2 835.12 2835119.00 180Loose 1 3
RR 5 3 925.02 3925024.00 180Loose 1 6
RR 5 1 866.70 1866701.00 180Loose 1 5
RR 5 2 983.80 2983802.00 180Loose 1 1
RR 5 1 878.14 1878143.00 180Loose 1 4
C 1 1 873.09 1873090.00 180Loose 1 2
C 1 1 899.35 1899349.00 180Loose 1 6
C 1 1 924.22 1924216.00 180Loose 1 5
```

```
C 1 2 842.02 2842018.00 180Loose 1 3
C 1 1 924.49 1924493.00 180Loose 1 1
C 1 1 966.00 1966000.00 180Loose 1 4
C 2 1 893.24 1893236.00 180Loose 1 2
C 2 2 868.79 2868793.00 180Loose 1 6
C 2 3 877.82 3877823.00 180Loose 1 3
C 2 2 886.94 2886943.00 180Loose 1 5
C 2 1 961.01 1961009.00 180Loose 1 1
C 2 1 979.17 1979167.00 180Loose 1 4
C 3 2 834.43 2834429.00 180Loose 1 2
C 3 2 898.36 2898364.00 180Loose 1 6
C 3 3 886.67 3886673.00 180Loose 1 5
C 3 1 951.29 1951286.00 180Loose 1 3
C 3 1 928.70 1928701.00 180Loose 1 1
C 3 1 892.40 1892398.00 180Loose 1 4
C 4 0 944.24 944238.00 180Loose 1 2
C 4 0 966.79 966794.00 180Loose 1 6
C 4 1 958.70 1958698.00 180Loose 1 5
C 4 0 927.68 927675.00 180Loose 1 3
C 4 1 896.39 1896390.00 180Loose 1 1
C 4 1 875.65 1875645.00 180Loose 1 4
C 5 1 958.41 1958412.00 180Loose 1 2
C 5 1 932.35 1932353.00 180Loose 1 6
C 5 1 933.92 1933924.00 180Loose 1 5
C 5 0 992.15 992147.00 180Loose 1 3
C 5 2 860.78 2860782.00 180Loose 1 1
C 5 1 931.30 1931304.00 180Loose 1 4
R 1 2 948.04 2948040.00 180Loose 1 2
R 1 2 886.02 2886015.00 180Loose 1 6
R 1 1 894.48 1894481.00 180Loose 1 5
R 1 1 927.59 1927588.00 180Loose 1 3
R 1 1 954.71 1954711.00 180Loose 1 1
R 1 1 955.96 1955962.00 180Loose 1 4
R 2 2 963.58 2963576.00 180Loose 1 2
R 2 1 946.68 1946675.00 180Loose 1 6
R 2 2 868.94 2868938.00 180Loose 1 5
R 2 2 851.91 2851906.00 180Loose 1 3
R 2 1 946.71 1946707.00 180Loose 1 1
R 2 0 881.38 881381.00 180Loose 1 4
R 3 2 987.14 2987144.00 180Loose 1 2
R 3 1 932.66 1932662.00 180Loose 1 6
R 3 2 900.93 2900932.00 180Loose 1 5
R 3 1 971.10 1971104.00 180Loose 1 3
R 3 2 908.45 2908447.00 180Loose 1 1
R 3 1 977.21 1977209.00 180Loose 1 4
R 4 1 949.60 1949603.00 180Loose 1 2
R 4 1 957.71 1957710.00 180Loose 1 6
R 4 1 883.12 1883123.00 180Loose 1 5
R 4 1 890.00 1889999.00 180Loose 1 3
R 4 2 990.12 2990118.00 180Loose 1 1
R 4 0 916.23 916233.00 180Loose 1 4
R 5 2 898.72 2898724.00 180Loose 1 2
R 5 3 985.82 3985824.00 180Loose 1 6
R 5 2 901.41 2901407.00 180Loose 1 5
R 5 1 984.74 1984736.00 180Loose 1 3
R 5 3 948.72 3948724.00 180Loose 1 1
R 5 2 923.76 2923764.00 180Loose 1 4
CR 1 1 867.67 1867666.00 180Loose 1 2
CR 1 1 940.63 1940634.00 180Loose 1 6
CR 1 1 887.58 1887581.00 180Loose 1 5
CR 1 2 885.18 2885180.00 180Loose 1 3
CR 1 0 852.45 852451.00 180Loose 1 1
CR 1 0 884.93 884931.00 180Loose 1 4
CR 2 1 927.04 1927040.00 180Loose 1 2
CR 2 2 983.11 2983105.00 180Loose 1 6
CR 2 1 978.08 1978083.00 180Loose 1 5
CR 2 2 888.27 2888272.00 180Loose 1 3
CR 2 3 821.04 3821042.00 180Loose 1 1
CR 2 2 978.78 2978777.00 180Loose 1 4
CR 3 0 895.65 895649.00 180Loose 1 2
CR 3 1 928.60 1928597.00 180Loose 1 6
CR 3 2 908.07 2908068.00 180Loose 1 5
CR 3 2 889.90 2889896.00 180Loose 1 3
CR 3 2 930.04 2930035.00 180Loose 1 1
CR 3 1 891.85 1891847.00 180Loose 1 4
CR 4 1 909.04 1909042.00 180Loose 1 2
CR 4 2 873.68 2873682.00 180Loose 1 6
CR 4 1 912.30 1912297.00 180Loose 1 5
CR 4 0 969.23 969226.00 180Loose 1 1
CR 4 0 964.06 964064.00 180Loose 1 3
CR 4 2 883.53 2883534.00 180Loose 1 4
CR 5 2 980.98 2980982.00 180Loose 1 2
CR 5 2 918.50 2918501.00 180Loose 1 6
CR 5 1 912.84 1912836.00 180Loose 1 5
CR 5 1 877.20 1877202.00 180Loose 1 3
CR 5 2 887.74 2887743.00 180Loose 1 1
CR 5 2 926.30 2926304.00 180Loose 1 4
RR 1 2 928.57 2928566.00 180Loose 1 0
RR 2 3 1008.32 4008323.00 180Loose 1 0
RR 3 1 921.76 1921756.00 180Loose 1 0
RR 4 1 946.42 1946423.00 180Loose 1 0
RR 5 2 1100.75 3100751.00 180Loose 1 0
C 1 3 825.42 3825421.00 180Loose 1 0
C 2 2 906.46 2906461.00 180Loose 1 0
C 3 1 1004.80 2004801.00 180Loose 1 0
C 4 1 931.95 1931950.00 180Loose 1 0
C 5 2 907.52 2907519.00 180Loose 1 0
R 1 1 925.50 1925500.00 180Loose 1 0
R 2 2 893.52 2893524.00 180Loose 1 0
R 3 1 916.56 1916555.00 180Loose 1 0
R 4 1 949.74 1949739.00 180Loose 1 0
R 5 1 920.88 1920882.00 180Loose 1 0
CR 1 2 874.54 2874540.00 180Loose 1 0
CR 2 2 889.95 2889951.00 180Loose 1 0
CR 3 3 895.98 3895975.00 180Loose 1 0
CR 4 1 940.96 1940962.00 180Loose 1 0
CR 5 2 872.38 2872378.00 180Loose 1 0
```

Online Stochastic Algorithm Solutions

```
Algorithm OnlineClass Unassigned Length Obj Instance InstanceNr
NN 1 2 1254.65 3254654.00 180Loose 1
NN 2 2 1274.82 3274815.00 180Loose 1
NN 3 2 1254.65 3254654.00 180Loose 1
NN 4 4 1237.25 5237253.00 180Loose 1
NN 5 4 1254.04 5254044.00 180Loose 1
NI 1 1 968.79 1968794.00 180Loose 1
NI 2 1 971.67 1971674.00 180Loose 1
NI 3 1 968.79 1968794.00 180Loose 1
NI 4 2 1036.26 3036262.00 180Loose 1
NI 5 2 1073.13 3073132.00 180Loose 1
LO 1 15 740.28 15740284.00 180Loose 1
LO 2 15 759.42 15759419.00 180Loose 1
LO 3 15 740.28 15740284.00 180Loose 1
LO 4 16 771.99 16771989.00 180Loose 1
LO 5 19 741.61 19741607.00 180Loose 1
```

A.5.2 180TIGHT

Offline solution

```
Algorithm OnlineClass Unassigned Length Obj Instance InstanceNr
OFF 0 0 959971.0 959971.0 180Tight 2
```

Oblivious Online Solutions

```
Algorithm OnlineClass Unassigned Length Obj Instance InstanceNr runNr
RR 1 3 961.30 3961297.00 180Tight 2 5
RR 1 2 1002.97 3002968.00 180Tight 2 2
RR 1 1 1019.52 2019517.00 180Tight 2 1
RR 1 1 955.75 1955751.00 180Tight 2 6
RR 1 0 1002.51 1002511.00 180Tight 2 4
RR 1 1 1004.25 2004254.00 180Tight 2 3
RR 2 2 1001.07 3001065.00 180Tight 2 5
RR 2 1 1046.43 2046430.00 180Tight 2 2
RR 2 2 958.60 2958603.00 180Tight 2 1
RR 2 1 972.84 1972838.00 180Tight 2 6
RR 2 3 925.04 3925035.00 180Tight 2 4
RR 2 2 952.83 2952830.00 180Tight 2 3
RR 3 1 1024.99 2024992.00 180Tight 2 5
RR 3 0 1081.53 1081528.00 180Tight 2 2
RR 3 1 1109.20 2109203.00 180Tight 2 1
RR 3 2 1068.41 3068409.00 180Tight 2 6
RR 3 1 955.25 1955254.00 180Tight 2 4
RR 3 0 1071.43 1071428.00 180Tight 2 3
RR 4 0 1009.34 1009339.00 180Tight 2 5
RR 4 1 1001.61 2001612.00 180Tight 2 2
RR 4 1 1037.99 2037987.00 180Tight 2 1
RR 4 1 1001.72 2001718.00 180Tight 2 6
RR 4 2 1010.10 3010098.00 180Tight 2 4
RR 4 0 1083.15 1083151.00 180Tight 2 3
RR 5 1 1085.27 2085271.00 180Tight 2 5
RR 5 0 989.74 989740.00 180Tight 2 2
RR 5 1 982.46 1982462.00 180Tight 2 1
RR 5 1 1007.98 2007984.00 180Tight 2 6
RR 5 0 1012.79 1012793.00 180Tight 2 4
RR 5 0 1007.69 1007692.00 180Tight 2 3
C 1 1 985.47 1985468.00 180Tight 2 5
C 1 0 978.00 978004.00 180Tight 2 2
C 1 0 989.07 989065.00 180Tight 2 1
C 1 0 1025.77 1025773.00 180Tight 2 4
C 1 1 998.13 1998131.00 180Tight 2 6
C 1 1 968.35 1968352.00 180Tight 2 3
C 2 1 969.48 1969483.00 180Tight 2 5
C 2 1 967.74 1967735.00 180Tight 2 2
C 2 0 964.78 964775.00 180Tight 2 1
C 2 2 973.37 2973370.00 180Tight 2 4
C 2 0 1025.77 1025771.00 180Tight 2 6
C 2 0 1032.52 1032520.00 180Tight 2 3
C 3 0 967.93 967932.00 180Tight 2 5
C 3 0 1012.41 1012407.00 180Tight 2 2
C 3 0 1047.93 1047930.00 180Tight 2 1
C 3 1 943.62 1943622.00 180Tight 2 6
C 3 0 1007.46 1007460.00 180Tight 2 4
C 4 0 992.30 992300.00 180Tight 2 5
```

```
C 4 0 990.44 990443.00 180Tight 2 2
C 4 0 990.83 990825.00 180Tight 2 1
C 4 0 1016.63 1016632.00 180Tight 2 4
C 3 0 990.16 990164.00 180Tight 2 3
C 4 0 1011.13 1011129.00 180Tight 2 6
C 5 0 1015.62 1015618.00 180Tight 2 5
C 5 0 978.34 978343.00 180Tight 2 2
C 5 1 950.52 1950515.00 180Tight 2 1
C 5 0 1011.43 1011434.00 180Tight 2 4
C 5 0 982.55 982550.00 180Tight 2 6
C 4 0 980.11 980110.00 180Tight 2 3
R 1 0 961.09 961090.00 180Tight 2 5
R 1 1 966.33 1966325.00 180Tight 2 2
R 1 1 945.74 1945743.00 180Tight 2 1
R 1 0 959.05 959050.00 180Tight 2 4
R 1 0 935.47 935467.00 180Tight 2 6
C 5 0 994.04 994042.00 180Tight 2 3
R 2 0 1011.06 1011060.00 180Tight 2 5
R 2 0 957.27 957271.00 180Tight 2 2
R 2 0 1017.83 1017834.00 180Tight 2 1
R 2 0 996.30 996295.00 180Tight 2 4
R 2 1 968.37 1968366.00 180Tight 2 6
R 1 0 1039.30 1039299.00 180Tight 2 3
R 3 0 1002.81 1002814.00 180Tight 2 5
R 3 0 1049.60 1049595.00 180Tight 2 2
R 3 0 976.18 976184.00 180Tight 2 1
R 3 0 989.64 989639.00 180Tight 2 4
R 3 1 967.11 1967106.00 180Tight 2 6
R 2 0 1047.30 1047303.00 180Tight 2 3
R 4 0 1011.13 1011129.00 180Tight 2 5
R 4 1 1011.08 2011078.00 180Tight 2 2
R 4 0 984.51 984510.00 180Tight 2 1
R 4 1 998.32 1998315.00 180Tight 2 4
R 4 1 955.94 1955942.00 180Tight 2 6
R 3 1 941.58 1941582.00 180Tight 2 3
R 5 1 965.73 1965729.00 180Tight 2 5
R 5 0 959.05 959050.00 180Tight 2 2
R 5 0 994.23 994233.00 180Tight 2 1
R 5 0 991.25 991251.00 180Tight 2 4
R 5 1 969.74 1969737.00 180Tight 2 6
R 4 0 1036.54 1036540.00 180Tight 2 3
CR 1 3 999.65 3999654.00 180Tight 2 5
CR 1 3 987.03 3987027.00 180Tight 2 2
CR 1 1 943.50 1943495.00 180Tight 2 1
CR 1 2 1038.97 3038974.00 180Tight 2 4
CR 1 1 1007.82 2007818.00 180Tight 2 6
R 5 1 972.85 1972845.00 180Tight 2 3
CR 2 2 991.12 2991124.00 180Tight 2 5
CR 2 2 1000.17 3000167.00 180Tight 2 2
CR 2 1 1072.97 2072968.00 180Tight 2 1
CR 2 3 1014.39 4014385.00 180Tight 2 4
CR 2 2 1034.63 3034625.00 180Tight 2 6
CR 1 2 1012.60 3012600.00 180Tight 2 3
CR 3 2 979.58 2979582.00 180Tight 2 5
CR 3 0 1031.96 1031963.00 180Tight 2 2
CR 3 2 951.18 2951178.00 180Tight 2 1
CR 3 1 1066.87 2066866.00 180Tight 2 4
CR 3 0 983.52 983516.00 180Tight 2 6
CR 4 1 973.36 1973360.00 180Tight 2 5
CR 2 3 968.46 3968455.00 180Tight 2 3
CR 4 1 1026.79 2026789.00 180Tight 2 2
CR 4 1 1041.03 2041033.00 180Tight 2 1
CR 4 1 1020.46 2020455.00 180Tight 2 4
CR 4 3 1052.41 4052413.00 180Tight 2 6
CR 5 2 951.47 2951466.00 180Tight 2 5
CR 3 2 995.00 2994996.00 180Tight 2 3
CR 5 3 913.71 3913705.00 180Tight 2 2
CR 5 1 961.16 1961158.00 180Tight 2 1
CR 5 1 1019.85 2019848.00 180Tight 2 4
CR 5 2 1047.68 3047681.00 180Tight 2 6
CR 4 1 1069.31 2069306.00 180Tight 2 3
CR 5 0 1025.57 1025569.00 180Tight 2 3
RR 1 1 999.30 1999304.00 180Tight 2 0
RR 2 1 1082.97 2082970.00 180Tight 2 0
RR 3 2 1006.76 3006761.00 180Tight 2 0
RR 4 2 1082.03 3082033.00 180Tight 2 0
RR 5 1 1026.87 2026867.00 180Tight 2 0
C 1 0 1008.67 1008673.00 180Tight 2 0
C 2 0 962.80 962796.00 180Tight 2 0
C 3 0 983.88 983878.00 180Tight 2 0
C 4 0 1063.38 1063384.00 180Tight 2 0
C 5 0 992.72 992722.00 180Tight 2 0
R 1 0 964.78 964775.00 180Tight 2 0
R 2 0 989.07 989065.00 180Tight 2 0
R 3 0 1011.35 1011353.00 180Tight 2 0
R 4 1 937.83 1937830.00 180Tight 2 0
R 5 0 995.36 995355.00 180Tight 2 0
CR 1 1 1000.25 2000251.00 180Tight 2 0
CR 2 2 1048.23 3048234.00 180Tight 2 0
CR 3 2 949.89 2949886.00 180Tight 2 0
CR 4 3 1032.34 4032341.00 180Tight 2 0
CR 5 1 1020.34 2020337.00 180Tight 2 0
```

Online Stochastic Algorithm Solutions

```
Algorithm OnlineClass Unassigned Length Obj Instance InstanceNr
NN 1 4 1237.58 5237582.00 180Tight 2
NN 2 4 1237.58 5237582.00 180Tight 2
NN 3 4 1237.58 5237582.00 180Tight 2
NN 4 4 1252.54 5252544.00 180Tight 2
NN 5 5 1269.99 6269992.00 180Tight 2
NI 1 1 1064.53 2064532.00 180Tight 2
NI 2 1 1064.53 2064532.00 180Tight 2
NI 3 1 1064.53 2064532.00 180Tight 2
NI 4 1 1064.53 2064532.00 180Tight 2
NI 5 2 1052.21 3052211.00 180Tight 2
LO 1 5 1008.20 6008195.00 180Tight 2
LO 2 5 1008.20 6008195.00 180Tight 2
LO 3 5 1008.20 6008195.00 180Tight 2
LO 4 6 1005.99 7005990.00 180Tight 2
LO 5 5 1028.02 6028020.00 180Tight 2
```

A.5.3 600LOOSE

Offline solution

```
Algorithm OnlineClass Unassigned Length Obj Instance InstanceNr
OFF 0 0 489621.0 489621.0 600Loose 3
```

Oblivious Online Solutions

```
Algorithm OnlineClass Unassigned Length Obj Instance InstanceNr runNr
RR 1 0 619.92 619917.00 600Loose 3 5
RR 1 0 680.46 680455.00 600Loose 3 4
RR 1 0 645.43 645432.00 600Loose 3 2
RR 1 0 746.71 746709.00 600Loose 3 1
RR 1 0 645.61 645611.00 600Loose 3 6
RR 1 0 613.91 613912.00 600Loose 3 3
RR 2 0 732.50 732498.00 600Loose 3 2
RR 2 0 655.12 655117.00 600Loose 3 6
RR 2 0 617.39 617387.00 600Loose 3 4
RR 2 0 655.01 655013.00 600Loose 3 5
RR 2 0 613.12 613117.00 600Loose 3 1
RR 2 0 662.20 662196.00 600Loose 3 3
RR 3 0 649.03 649031.00 600Loose 3 2
RR 3 1 762.92 1762924.00 600Loose 3 4
RR 3 0 783.94 783941.00 600Loose 3 6
RR 3 0 651.45 651448.00 600Loose 3 5
RR 3 0 628.06 628058.00 600Loose 3 1
RR 3 0 659.76 659762.00 600Loose 3 3
RR 4 0 640.43 640428.00 600Loose 3 2
RR 4 0 629.56 629555.00 600Loose 3 4
RR 4 0 662.92 662918.00 600Loose 3 5
RR 4 0 678.79 678786.00 600Loose 3 1
RR 4 0 631.64 631643.00 600Loose 3 6
RR 4 0 657.31 657314.00 600Loose 3 3
RR 5 0 645.06 645061.00 600Loose 3 2
RR 5 0 697.69 697688.00 600Loose 3 4
RR 5 0 695.62 695623.00 600Loose 3 5
RR 5 0 745.35 745349.00 600Loose 3 6
RR 5 0 696.06 696055.00 600Loose 3 1
RR 5 0 650.25 650250.00 600Loose 3 3
C 1 0 555.93 555927.00 600Loose 3 2
C 1 0 572.53 572533.00 600Loose 3 4
C 1 0 529.17 529169.00 600Loose 3 5
C 1 0 542.77 542771.00 600Loose 3 6
C 1 0 507.92 507922.00 600Loose 3 1
C 1 0 505.20 505201.00 600Loose 3 3
C 2 0 534.22 534220.00 600Loose 3 2
C 2 0 574.18 574180.00 600Loose 3 4
C 2 0 590.25 590248.00 600Loose 3 5
C 2 0 586.49 586486.00 600Loose 3 6
C 2 0 542.92 542917.00 600Loose 3 1
C 3 0 551.50 551495.00 600Loose 3 2
C 3 0 540.97 540972.00 600Loose 3 5
C 3 0 536.63 536632.00 600Loose 3 4
C 3 0 525.19 525188.00 600Loose 3 6
C 2 0 561.86 561859.00 600Loose 3 3
C 3 0 562.31 562307.00 600Loose 3 1
C 4 0 674.07 674065.00 600Loose 3 2
C 4 0 633.83 633834.00 600Loose 3 5
C 4 0 570.92 570919.00 600Loose 3 4
C 4 0 610.54 610540.00 600Loose 3 6
C 4 0 590.03 590026.00 600Loose 3 1
C 3 0 552.62 552622.00 600Loose 3 3
C 5 0 755.31 755308.00 600Loose 3 5
C 5 2 735.43 2735427.00 600Loose 3 2
C 5 0 645.30 645297.00 600Loose 3 4
C 5 1 722.67 1722669.00 600Loose 3 6
C 5 0 615.68 615680.00 600Loose 3 1
C 4 0 592.32 592319.00 600Loose 3 3
R 1 0 680.13 680125.00 600Loose 3 5
R 1 0 607.69 607691.00 600Loose 3 2
R 1 0 555.79 555794.00 600Loose 3 4
R 1 0 671.47 671465.00 600Loose 3 6
```

```
R 1 0 623.96 623964.00 600Loose 3 1
C 5 0 702.72 702724.00 600Loose 3 3
R 2 0 732.11 732109.00 600Loose 3 5
R 2 0 689.10 689100.00 600Loose 3 4
R 2 0 613.51 613512.00 600Loose 3 2
R 2 0 676.47 676468.00 600Loose 3 6
R 2 0 610.24 610240.00 600Loose 3 1
R 3 0 645.73 645732.00 600Loose 3 5
R 3 0 651.10 651104.00 600Loose 3 4
R 3 0 677.34 677335.00 600Loose 3 2
R 3 0 666.84 666836.00 600Loose 3 6
R 1 0 707.71 707708.00 600Loose 3 3
R 3 0 596.38 596382.00 600Loose 3 1
R 4 0 648.18 648175.00 600Loose 3 5
R 4 0 694.79 694787.00 600Loose 3 4
R 4 0 717.55 717547.00 600Loose 3 2
R 4 0 659.02 659018.00 600Loose 3 6
R 2 0 671.79 671788.00 600Loose 3 3
R 4 0 657.91 657914.00 600Loose 3 1
R 5 0 661.00 660998.00 600Loose 3 5
R 5 3 777.36 3777357.00 600Loose 3 4
R 5 0 738.29 738293.00 600Loose 3 2
R 5 1 719.56 1719560.00 600Loose 3 6
R 3 0 633.66 633664.00 600Loose 3 3
R 5 2 736.65 2736650.00 600Loose 3 1
CR 1 0 588.23 588225.00 600Loose 3 5
CR 1 0 550.79 550790.00 600Loose 3 4
CR 1 0 584.02 584024.00 600Loose 3 2
CR 1 0 615.17 615167.00 600Loose 3 6
CR 1 0 597.04 597035.00 600Loose 3 1
R 4 0 746.09 746088.00 600Loose 3 3
CR 2 0 618.11 618109.00 600Loose 3 5
CR 2 0 602.98 602976.00 600Loose 3 4
CR 2 0 610.85 610853.00 600Loose 3 2
CR 2 0 611.00 610999.00 600Loose 3 6
CR 2 0 587.65 587654.00 600Loose 3 1
R 5 0 720.85 720852.00 600Loose 3 3
CR 3 0 565.88 565876.00 600Loose 3 5
CR 3 0 586.66 586663.00 600Loose 3 4
CR 3 0 569.20 569195.00 600Loose 3 2
CR 3 0 586.97 586966.00 600Loose 3 6
CR 3 0 574.15 574151.00 600Loose 3 1
CR 1 1 606.82 1606816.00 600Loose 3 3
CR 4 0 643.63 643627.00 600Loose 3 5
CR 4 0 646.13 646131.00 600Loose 3 4
CR 4 0 688.10 688101.00 600Loose 3 2
CR 4 0 654.22 654223.00 600Loose 3 6
CR 4 0 646.06 646064.00 600Loose 3 1
CR 2 0 617.04 617037.00 600Loose 3 3
CR 5 0 683.18 683176.00 600Loose 3 5
CR 5 0 619.72 619720.00 600Loose 3 4
CR 5 0 617.28 617275.00 600Loose 3 2
CR 5 1 647.23 1647234.00 600Loose 3 6
CR 5 0 618.13 618130.00 600Loose 3 1
CR 3 0 603.62 603623.00 600Loose 3 3
CR 4 0 605.42 605415.00 600Loose 3 3
CR 5 0 706.06 706064.00 600Loose 3 3
RR 1 0 586.00 585999.00 600Loose 3 0
RR 2 0 669.22 669217.00 600Loose 3 0
RR 3 0 697.29 697286.00 600Loose 3 0
RR 4 0 647.88 647879.00 600Loose 3 0
RR 5 0 653.11 653111.00 600Loose 3 0
```

```
C 1 0 507.92 507922.00 600Loose 3 0
C 2 0 596.84 596840.00 600Loose 3 0
C 3 0 525.40 525403.00 600Loose 3 0
C 4 0 587.46 587459.00 600Loose 3 0
C 5 1 681.45 1681452.00 600Loose 3 0
R 1 0 575.61 575610.00 600Loose 3 0
R 2 0 669.13 669131.00 600Loose 3 0
R 3 0 754.33 754329.00 600Loose 3 0
R 4 0 641.60 641595.00 600Loose 3 0
R 5 1 719.44 1719439.00 600Loose 3 0
CR 1 0 594.71 594705.00 600Loose 3 0
CR 2 0 602.86 602856.00 600Loose 3 0
CR 3 0 613.60 613602.00 600Loose 3 0
CR 4 0 612.02 612018.00 600Loose 3 0
CR 5 1 719.13 1719134.00 600Loose 3 0
```

Online Stochastic Algorithm Solutions

```
Algorithm OnlineClass Unassigned Length Obj Instance InstanceNr
NN 1 1 1312.34 2312340.00 600Loose 3
NN 2 1 1298.47 2298473.00 600Loose 3
NN 3 1 1298.47 2298473.00 600Loose 3
NN 4 1 1281.46 2281455.00 600Loose 3
NN 5 1 1495.67 2495668.00 600Loose 3
NI 1 0 670.05 670049.00 600Loose 3
NI 2 0 721.48 721484.00 600Loose 3
NI 3 0 764.08 764080.00 600Loose 3
NI 4 0 705.22 705219.00 600Loose 3
NI 5 0 745.94 745942.00 600Loose 3
LO 1 3 568.27 3568268.00 600Loose 3
LO 2 4 565.09 4565094.00 600Loose 3
LO 3 4 561.96 4561961.00 600Loose 3
LO 4 7 727.12 7727115.00 600Loose 3
LO 5 7 610.32 7610320.00 600Loose 3
```

A.5.4 600TIGHT

Offline solution

```
Algorithm OnlineClass Unassigned Length Obj Instance InstanceNr
OFF 0 0 638965.0 638965.0 600Tight 4
```

Oblivious Online Solutions

```
Algorithm OnlineClass Unassigned Length Obj Instance InstanceNr runNr
RR 1 2 748.45 2748449.00 600Tight 4 2
RR 1 0 772.43 772430.00 600Tight 4 5
RR 1 0 788.02 788018.00 600Tight 4 3
RR 1 1 749.26 1749259.00 600Tight 4 1
RR 1 0 768.46 768461.00 600Tight 4 6
RR 1 1 767.52 1767515.00 600Tight 4 4
RR 2 0 756.14 756139.00 600Tight 4 2
RR 2 1 771.66 1771660.00 600Tight 4 5
RR 2 2 725.64 2725641.00 600Tight 4 1
RR 2 0 743.41 743409.00 600Tight 4 6
RR 2 0 766.05 766045.00 600Tight 4 3
RR 2 0 728.69 728694.00 600Tight 4 4
RR 3 0 758.19 758192.00 600Tight 4 1
RR 3 3 760.41 3760413.00 600Tight 4 5
RR 3 0 789.85 789854.00 600Tight 4 2
RR 3 0 841.62 841617.00 600Tight 4 6
RR 3 1 901.53 1901532.00 600Tight 4 3
RR 3 1 829.06 1829056.00 600Tight 4 4
RR 4 0 887.79 887792.00 600Tight 4 1
RR 4 0 700.91 700905.00 600Tight 4 2
RR 4 0 790.20 790197.00 600Tight 4 5
RR 4 1 761.08 1761080.00 600Tight 4 6
RR 4 1 812.23 1812233.00 600Tight 4 3
RR 4 1 862.11 1862110.00 600Tight 4 4
RR 5 0 821.71 821706.00 600Tight 4 1
RR 5 0 775.57 775571.00 600Tight 4 2
RR 5 0 805.91 805914.00 600Tight 4 5
RR 5 0 872.65 872646.00 600Tight 4 6
RR 5 0 793.18 793181.00 600Tight 4 3
RR 5 1 836.37 1836365.00 600Tight 4 4
C 1 0 728.05 728052.00 600Tight 4 1
C 1 0 761.74 761744.00 600Tight 4 2
C 1 1 732.08 1732078.00 600Tight 4 5
C 1 0 782.28 782276.00 600Tight 4 6
C 1 0 730.76 730764.00 600Tight 4 3
C 1 0 762.79 762785.00 600Tight 4 4
C 2 0 754.10 754100.00 600Tight 4 1
C 2 0 758.66 758657.00 600Tight 4 2
C 2 0 802.60 802599.00 600Tight 4 5
C 2 0 879.25 879246.00 600Tight 4 6
C 2 2 808.68 2808679.00 600Tight 4 3
C 2 1 913.29 1913293.00 600Tight 4 4
C 3 0 774.34 774336.00 600Tight 4 1
C 3 0 758.92 758924.00 600Tight 4 2
C 3 0 742.43 742428.00 600Tight 4 5
C 3 0 723.53 723527.00 600Tight 4 6
C 3 0 767.39 767393.00 600Tight 4 3
C 3 0 884.54 884539.00 600Tight 4 4
C 4 2 743.12 2743120.00 600Tight 4 1
C 4 0 709.45 709449.00 600Tight 4 2
C 4 1 742.26 1742255.00 600Tight 4 5
C 4 1 744.33 1744333.00 600Tight 4 6
C 4 0 765.78 765777.00 600Tight 4 3
C 4 1 735.94 1735943.00 600Tight 4 4
C 5 3 858.72 3858715.00 600Tight 4 1
C 5 2 843.88 2843877.00 600Tight 4 2
C 5 1 745.46 1745463.00 600Tight 4 6
C 5 1 813.70 1813697.00 600Tight 4 5
C 5 1 823.01 1823005.00 600Tight 4 3
C 5 1 895.38 1895377.00 600Tight 4 4
R 1 0 849.21 849205.00 600Tight 4 2
R 1 0 820.89 820890.00 600Tight 4 1
R 1 1 742.82 1742820.00 600Tight 4 6
R 1 0 738.17 738166.00 600Tight 4 5
R 1 1 865.60 1865600.00 600Tight 4 3
R 1 1 789.42 1789416.00 600Tight 4 4
R 2 0 829.82 829818.00 600Tight 4 1
R 2 0 779.16 779157.00 600Tight 4 2
R 2 1 749.95 1749952.00 600Tight 4 6
R 2 1 807.70 1807702.00 600Tight 4 3
R 2 0 784.69 784686.00 600Tight 4 5
R 2 1 821.33 1821326.00 600Tight 4 4
R 3 1 755.99 1755993.00 600Tight 4 1
R 3 0 733.16 733157.00 600Tight 4 2
R 3 0 748.24 748244.00 600Tight 4 6
R 3 0 784.40 784395.00 600Tight 4 5
R 3 2 739.54 2739541.00 600Tight 4 3
R 3 0 781.41 781414.00 600Tight 4 4
```

```
R 4 1 724.01 1724006.00 600Tight 4 1
R 4 2 790.89 2790891.00 600Tight 4 2
R 4 0 752.82 752816.00 600Tight 4 6
R 4 1 751.21 1751212.00 600Tight 4 5
R 4 0 741.64 741643.00 600Tight 4 3
R 4 1 792.68 1792675.00 600Tight 4 4
R 5 2 770.69 2770689.00 600Tight 4 1
R 5 1 821.46 1821459.00 600Tight 4 2
R 5 1 835.35 1835354.00 600Tight 4 6
R 5 2 846.77 2846773.00 600Tight 4 5
R 5 2 863.85 2863846.00 600Tight 4 3
R 5 4 759.57 4759574.00 600Tight 4 4
CR 1 0 714.80 714801.00 600Tight 4 1
CR 1 1 761.55 1761552.00 600Tight 4 2
CR 1 0 777.79 777787.00 600Tight 4 6
CR 1 0 753.41 753408.00 600Tight 4 5
CR 1 0 847.83 847830.00 600Tight 4 3
CR 1 0 907.43 907428.00 600Tight 4 4
CR 2 1 790.47 1790471.00 600Tight 4 1
CR 2 0 739.28 739275.00 600Tight 4 2
CR 2 1 818.21 1818211.00 600Tight 4 6
CR 2 2 716.53 2716533.00 600Tight 4 5
CR 2 1 785.20 1785201.00 600Tight 4 3
CR 2 0 917.57 917569.00 600Tight 4 4
CR 3 2 788.51 2788509.00 600Tight 4 1
CR 3 0 752.04 752038.00 600Tight 4 2
CR 3 0 772.89 772888.00 600Tight 4 6
CR 3 0 755.25 755252.00 600Tight 4 5
CR 3 0 844.83 844827.00 600Tight 4 3
CR 3 0 840.43 840428.00 600Tight 4 4
CR 4 1 784.26 1784259.00 600Tight 4 1
CR 4 1 760.64 1760643.00 600Tight 4 2
CR 4 0 750.95 750947.00 600Tight 4 6
CR 4 0 725.25 725245.00 600Tight 4 5
CR 4 0 783.29 783291.00 600Tight 4 3
CR 4 0 840.64 840642.00 600Tight 4 4
CR 5 0 811.20 811196.00 600Tight 4 1
CR 5 0 789.27 789266.00 600Tight 4 2
CR 5 0 801.21 801206.00 600Tight 4 6
CR 5 0 890.07 890065.00 600Tight 4 3
CR 5 0 799.64 799640.00 600Tight 4 5
CR 5 0 835.07 835065.00 600Tight 4 4
RR 1 0 819.95 819949.00 600Tight 4 0
RR 2 0 804.85 804852.00 600Tight 4 0
RR 3 2 765.13 2765134.00 600Tight 4 0
RR 4 1 789.17 1789173.00 600Tight 4 0
RR 5 1 831.47 1831467.00 600Tight 4 0
C 1 1 731.18 1731181.00 600Tight 4 0
C 2 0 827.92 827922.00 600Tight 4 0
C 3 0 746.90 746901.00 600Tight 4 0
C 4 0 744.99 744988.00 600Tight 4 0
C 5 2 814.75 2814750.00 600Tight 4 0
R 1 0 771.92 771923.00 600Tight 4 0
R 2 1 837.04 1837041.00 600Tight 4 0
R 3 0 742.98 742976.00 600Tight 4 0
R 4 0 794.62 794619.00 600Tight 4 0
R 5 1 747.39 1747388.00 600Tight 4 0
CR 1 0 849.79 849790.00 600Tight 4 0
CR 2 2 791.90 2791900.00 600Tight 4 0
CR 3 0 768.59 768593.00 600Tight 4 0
CR 4 1 919.82 1919824.00 600Tight 4 0
CR 5 0 1012.95 1012954.00 600Tight 4 0
```

Online Stochastic Algorithm Solutions

```
Algorithm OnlineClass Unassigned Length Obj Instance InstanceNr
NN 1 1 1413.34 2413340.00 600Tight 4
NN 2 1 1440.08 2440079.00 600Tight 4
NN 3 1 1440.08 2440079.00 600Tight 4
NN 4 1 1433.34 2433339.00 600Tight 4
NN 5 1 1490.69 2490689.00 600Tight 4
NI 1 1 840.00 1839995.00 600Tight 4
NI 2 1 849.55 1849547.00 600Tight 4
NI 3 1 849.55 1849547.00 600Tight 4
NI 4 1 835.45 1835452.00 600Tight 4
NI 5 0 749.79 749789.00 600Tight 4
LO 1 3 754.95 3754948.00 600Tight 4
LO 2 3 765.70 3765696.00 600Tight 4
LO 3 3 765.70 3765696.00 600Tight 4
LO 4 2 716.31 2716310.00 600Tight 4
LO 5 5 808.00 5808004.00 600Tight 4
```

B Race Output

B.1 Impact Parameter Tuning - R Output

```
Racing methods for the selection of the best
Copyright (C) 2003 Mauro Birattari
This software comes with ABSOLUTELY NO WARRANTY

Race name....................Tuning of Impact Parameters on
                                                 class-solomon2
Number of candidates.........................................22
Number of available tasks....................................56
Max number of experiments..................................1200
Statistical test.................................Friedman test
Tasks seen before discarding.................................10
Initialization function......................................ok
Parallel Virtual Machine.....................................no


                            Markers:
                               x No test is performed.
                               - The test is performed and
                                 some candidates are discarded.
                               = The test is performed but
                                 no candidate is discarded.


+-+-----------+-----------+-----------+-----------+-----------+
| |       Task|      Alive|       Best|  Mean best| Exp so far|
+-+-----------+-----------+-----------+-----------+-----------+
|x|          1|         22|          4|  4.002e+09|         22|
|x|          2|         22|          4|  3.002e+09|         44|
|x|          3|         22|          4|  5.002e+09|         66|
|x|          4|         22|          4|  1.225e+10|         88|
|x|          5|         22|          4|    1.2e+10|        110|
|x|          6|         22|          4|  1.117e+10|        132|
|x|          7|         22|          4|    1.2e+10|        154|
|x|          8|         22|          4|   1.35e+10|        176|
|x|          9|         22|          4|    1.2e+10|        198|
|-|         10|         19|          4|   1.08e+10|        220|
|=|         11|         19|          4|   9.82e+09|        239|
|=|         12|         19|          4|  9.002e+09|        258|
|=|         13|         19|          4|   8.31e+09|        277|
|=|         14|         19|          4|  7.716e+09|        296|
|=|         15|         19|          4|  7.669e+09|        315|
|=|         16|         19|          4|   7.19e+09|        334|
|=|         17|         19|          4|  6.767e+09|        353|
|=|         18|         19|          4|  6.391e+09|        372|
|=|         19|         19|          4|  6.055e+09|        391|
|=|         20|         19|          4|  5.752e+09|        410|
|=|         21|         19|          4|  6.002e+09|        429|
|=|         22|         19|          4|  5.729e+09|        448|
|=|         23|         19|          4|  6.002e+09|        467|
|=|         24|         19|          4|  5.877e+09|        486|
|=|         25|         19|          4|  5.922e+09|        505|
|=|         26|         19|          4|  6.002e+09|        524|
|=|         27|         19|          4|  5.891e+09|        543|
|=|         28|         19|          4|  5.681e+09|        562|
|=|         29|         19|          4|  5.485e+09|        581|
|=|         30|         19|          4|  5.302e+09|        600|
|=|         31|         19|          4|  5.131e+09|        619|
|=|         32|         19|          4|  4.971e+09|        638|
|=|         33|         19|          4|  4.851e+09|        657|
|=|         34|         19|          4|  4.708e+09|        676|
|=|         35|         19|          4|  4.631e+09|        695|
|=|         36|         19|          4|   4.53e+09|        714|
|=|         37|         19|          4|  4.435e+09|        733|
|=|         38|         19|          4|  4.318e+09|        752|
|=|         39|         19|          4|  4.694e+09|        771|
|=|         40|         19|          4|  4.727e+09|        790|
|=|         41|         19|          4|  4.709e+09|        809|
|=|         42|         19|          4|  5.002e+09|        828|
|=|         43|         19|          4|  5.397e+09|        847|
|=|         44|         19|          4|  5.388e+09|        866|
|=|         45|         19|          4|  5.513e+09|        885|
|=|         46|         19|          4|  5.763e+09|        904|
|=|         47|         19|          4|  6.045e+09|        923|
|=|         48|         19|          4|  6.065e+09|        942|
|=|         49|         19|          4|  5.961e+09|        961|
|=|         50|         19|          4|  5.862e+09|        980|
|=|         51|         19|          4|  5.747e+09|        999|
|=|         52|         19|          4|  5.829e+09|       1018|
|=|         53|         19|          4|  5.964e+09|       1037|
|=|         54|         19|          4|  5.965e+09|       1056|
|=|         55|         19|          4|   6.02e+09|       1075|
|=|         56|         19|          4|  5.948e+09|       1094|
+-+-----------+-----------+-----------+-----------+-----------+

Selected candidate:           4        mean value:  5.948e+09

Description of the selected candidate:
  label               command
4 1-1-8 runImpact.jar 10 10 80


$precis
[1] "\nRacing methods for the selection of the best\nCopyright (C) 2003 Mauro Birattari\nThis software comes with ABSOLUTELY NO WARRA

$results
             [,1]        [,2]        [,3]        [,4]        [,5]        [,6]
 [1,] 30002666743 30002666743 10002124943  4002079342  4002079342  4002079342
 [2,] 35002573675 35002573675 14002267462  2002031084  2002031084  2002031084
 [3,] 43002223661 43002223661 34002147180  9001820688  9001820688  9001820688
 [4,] 52001876951 52001876951 46001794378 34001679408 34001679408 34001679408
 [5,] 26002466616 26002466616 17002207186 11002232000 11002232000 11002232000
 [6,] 28002266458 28002266458 27002037508  7001872851  7001872851  7001872851
 [7,] 40002179539 40002179539 29002035582 17001835444 17001835444 17001835444
 [8,] 52002107617 52002107617 42001922064 24001762993 24001762993 24001762993
 [9,]  5002838830  5002838830  1002506278     2490933     2490933     2490933
[10,]  8002504552  8002504552  2002456302     2147668     2147668     2147668
[11,]          NA          NA          NA     2105298     2105298     2105298
[12,]          NA          NA          NA     1864733     1864733     1864733
[13,]          NA          NA          NA     2295745     2295745     2295745
[14,]          NA          NA          NA     2334613     2334613     2334613
[15,]          NA          NA          NA  7002054322  7002054322  7002054322
[16,]          NA          NA          NA     1912129     1912129     1912129
[17,]          NA          NA          NA     2506797     2506797     2506797
[18,]          NA          NA          NA     2667283     2667283     2667283
[19,]          NA          NA          NA     2385567     2385567     2444337
[20,]          NA          NA          NA     1297031     1297031     1297031
[21,]          NA          NA          NA 11001971079 11001971079 11001971079
[22,]          NA          NA          NA     2259683     2259683     2259683
[23,]          NA          NA          NA 12002017015 12002017015 12002017015
[24,]          NA          NA          NA  3001088451  3001088451  3001088451
[25,]          NA          NA          NA  7001551941  7001551941  7001551941
[26,]          NA          NA          NA  8001706941  8001706941  8001706941
[27,]          NA          NA          NA  3001195437  3001195437  3001195437
[28,]          NA          NA          NA     2498030     2498030     2498030
[29,]          NA          NA          NA     2478148     2478148     2478148
[30,]          NA          NA          NA     2554385     2554385     2554385
[31,]          NA          NA          NA     2345474     2345474     2345474
[32,]          NA          NA          NA     2750943     2750943     2753417
[33,]          NA          NA          NA  1002597886  1002597886  1002597886
[34,]          NA          NA          NA     3077338     3077338     3077338
[35,]          NA          NA          NA  2002755043  2002755043  2002755043
[36,]          NA          NA          NA  1002004219  1002004219  1002004219
[37,]          NA          NA          NA  1001967221  1001967221  1001965344
[38,]          NA          NA          NA     1797627     1797627     1797627
[39,]          NA          NA          NA 19001612333 19001612333 19001612333
[40,]          NA          NA          NA  6001853616  6001853616  6001853616
[41,]          NA          NA          NA  4001882318  4001882318  4001882318
[42,]          NA          NA          NA 17001594059 17001594059 17001594059
[43,]          NA          NA          NA 22001478282 22001478282 22001478282
[44,]          NA          NA          NA  5001772265  5001772265  5001772265
[45,]          NA          NA          NA 11001679067 11001679067 11001679067
[46,]          NA          NA          NA 17001582342 17001582342 17001582342
[47,]          NA          NA          NA 19001574116 19001574116 19001574116
[48,]          NA          NA          NA  7001245393  7001245393  7001245393
[49,]          NA          NA          NA  1002073182  1002073182  1002073182
[50,]          NA          NA          NA  1002536319  1002536319  1002536319
[51,]          NA          NA          NA     1978379     1978379     1978379
[52,]          NA          NA          NA 10001774925 10001774925 10001774925
[53,]          NA          NA          NA 13002153736 13002153736 13002153736
[54,]          NA          NA          NA  6001878723  6001878723  6001878723
[55,]          NA          NA          NA  9002423915  9002423915  9002423915
[56,]          NA          NA          NA  2001923778  2001923778  2001923778
             [,7]        [,8]        [,9]       [,10]       [,11]       [,12]
 [1,]  4002079342  4002079342  4002079342  4002079342  4002079342  4002079342
 [2,]  2002031084  2002031084  2002031084  2002031084  2002031084  2002031084
 [3,]  9001820688  9001820688  9001820688  9001820688  9001820688  9001820688
 [4,] 34001679408 34001679408 34001679408 34001679408 34001679408 34001679408
 [5,] 11002232000 11002232000 11002232000 11002232000 11002232000 11002232000
 [6,]  7001872851  7001872851  7001872851  7001872851  7001872851  7001872851
 [7,] 17001835444 17001835444 17001835444 17001835444 17001835444 17001835444
 [8,] 24001762993 24001762993 24001762993 24001762993 24001762993 24001762993
 [9,]     2490933     2490933     2490933     2490933     2490933     2490933
[10,]     2147668     2147668     2147668     2147668     2147668     2147668
[11,]     2105298     2105298     2105298     2105298     2105298     2105298
[12,]     1864733     1864733     1864733     1864733     1864733     1864733
[13,]     2295745     2295745     2295745     2295745     2295745     2295745
[14,]     2334613     2334613     2334613     2334613     2334613     2334613
[15,]  7002054322  7002054322  7002054322  7002054322  7002054322  7002054322
[16,]     1912129     1912129     1912129     1912129     1912129     1912129
[17,]     2506797     2506797     2506797     2506797     2506797     2506797
[18,]     2667283     2667283     2667283     2667283     2667283     2667283
[19,]     2385567     2385567     2444337     2385567     2385567     2385567
[20,]     1297031     1297031     1297031     1297031     1297031     1297031
[21,] 11001971079 11001971079 11001971079 11001971079 11001971079 11001971079
[22,]     2259683     2259683     2259683     2259683     2259683     2259683
[23,] 12002017015 12002017015 12002017015 12002017015 12002017015 12002017015
[24,]  3001088451  3001088451  3001088451  3001088451  3001088451  3001088451
[25,]  7001551941  7001551941  7001551941  7001551941  7001551941  7001551941
[26,]  8001706941  8001706941  8001706941  8001706941  8001706941  8001706941
[27,]  3001195437  3001195437  3001195437  3001195437  3001195437  3001195437
[28,]     2498030     2498030     2498030     2498030     2498030     2498030
[29,]     2478148     2478148     2478148     2478148     2478148     2478148
[30,]     2554385     2554385     2554385     2554385     2554385     2554385
[31,]     2345474     2345474     2345474     2345474     2345474     2345474
[32,]     2750943     2750943     2750943     2750943     2750943     2750943
[33,]  1002597886  1002597886  1002597886  1002597886  1002597886  1002597886
[34,]     3077338     3077338     3077338     3077338     3077338     3077338
[35,]  2002755043  2002755043  2002755043  2002755043  2002755043  2002755043
[36,]  1002004219  1002004219  1002004219  1002004219  1002004219  1002004219
[37,]  1001967221  1001967221  1001967221  1001967221  1001967221  1001967221
[38,]     1797627     1797627     1797627     1797627     1797627     1797627
[39,] 19001612333 19001612333 19001612333 19001612333 19001612333 19001612333
[40,]  6001853616  6001853616  6001853616  6001853616  6001853616  6001853616
[41,]  4001882318  4001882318  4001882318  4001882318  4001882318  4001882318
[42,] 17001594059 17001594059 17001594059 17001594059 17001594059 17001594059
[43,] 22001478282 22001478282 22001478282 22001478282 22001478282 22001478282
[44,]  5001772265  5001772265  5001772265  5001772265  5001772265  5001772265
[45,] 11001679067 11001679067 11001679067 11001679067 11001679067 11001679067
[46,] 17001582342 17001582342 17001582342 17001582342 17001582342 17001582342
[47,] 19001574116 19001574116 19001574116 19001574116 19001574116 19001574116
[48,]  7001245393  7001245393  7001245393  7001245393  7001245393  7001245393
[49,]  1002073182  1002073182  1002073182  1002073182  1002073182  1002073182
[50,]  1002536319  1002536319  1002536319  1002536319  1002536319  1002536319
[51,]     1978379     1978379     1978379     1978379     1978379     1978379
[52,] 10001774925 10001774925 10001774925 10001774925 10001774925 10001774925
[53,] 13002153736 13002153736 13002153736 13002153736 13002153736 13002153736
[54,]  6001878723  6001878723  6001878723  6001878723  6001878723  6001878723
[55,]  9002423915  9002423915  9002423915  9002423915  9002423915  9002423915
[56,]  2001923778  2001923778  2001923778  2001923778  2001923778  2001923778
            [,13]       [,14]       [,15]       [,16]       [,17]       [,18]
 [1,]  4002079342  4002079342  4002079342  4002079342  4002079342  4002079342
 [2,]  2002031084  2002031084  2002031084  2002031084  2002031084  2002031084
 [3,]  9001820688  9001820688  9001820688  9001820688  9001820688  9001820688
 [4,] 34001679408 34001679408 34001679408 34001679408 34001679408 34001679408
 [5,] 11002232000 11002232000 11002232000 11002232000 11002232000 11002232000
 [6,]  7001872851  7001872851  7001872851  7001872851  7001872851  7001872851
 [7,] 17001835444 17001835444 17001835444 17001835444 17001835444 17001835444
 [8,] 24001762993 24001762993 24001762993 24001762993 24001762993 24001762993
 [9,]     2490933     2490933     2490933     2490933     2490933     2490933
[10,]     2147668     2147668     2147668     2147668     2147668     2147668
[11,]     2105298     2105298     2105298     2105298     2105298     2105298
[12,]     1864733     1864733     1864733     1864733     1864733     1864733
[13,]     2295745     2295745     2295745     2295745     2295745     2295745
[14,]     2334613     2334613     2334613     2334613     2334613     2334613
[15,]  7002054322  7002054322  7002054322  7002054322  7002054322  7002054322
[16,]     1912129     1912129     1912129     1912129     1912129     1912129
[17,]     2506797     2506797     2506797     2506797     2506797     2506797
[18,]     2667283     2667283     2667283     2667283     2667283     2667283
[19,]     2444337     2444337     2385567     2385567     2444337     2444337
[20,]     1297031     1297031     1297031     1297031     1297031     1297031
[21,] 11001971079 11001971079 11001971079 11001971079 11001971079 11001971079
[22,]     2259683     2259683     2259683     2259683     2259683     2259683
[23,] 12002017015 12002017015 12002017015 12002017015 12002017015 12002017015
[24,]  3001088451  3001088451  3001088451  3001088451  3001088451  3001088451
[25,]  7001551941  7001551941  7001551941  7001551941  7001551941  7001551941
[26,]  8001706941  8001706941  8001706941  8001706941  8001706941  8001706941
[27,]  3001195437  3001195437  3001195437  3001195437  3001195437  3001195437
[28,]     2498030     2498030     2498030     2498030     2498030     2498030
```

```
[29,]     2478148     2478148     2478148     2478148     2478148     2478148
[30,]     2554385     2554385     2554385     2554385     2554385     2554385
[31,]     2345474     2345474     2345474     2345474     2345474     2345474
[32,]     2750943     2750943     2750943     2750943     2750943     2750943
[33,]  1002597886  1002597886  1002597886  1002597886  1002597886  1002597886
[34,]     3077338     3077338     3077338     3077338     3077338     3077338
[35,]  2002755043  2002755043  2002755043  2002755043  2002755043  2002755043
[36,]  1002004219  1002004219  1002004219  1002004219  1002004219  1002004219
[37,]  1001967221  1001967221  1001967221  1001967221  1001967221  1001967221
[38,]     1797627     1797627     1797627     1797627     1797627     1797627
[39,] 19001612333 19001612333 19001612333 19001612333 19001612333 19001612333
[40,]  6001853616  6001853616  6001853616  6001853616  6001853616  6001853616
[41,]  4001882318  4001882318  4001882318  4001882318  4001882318  4001882318
[42,] 17001594059 17001594059 17001594059 17001594059 17001594059 17001594059
[43,] 22001478282 22001478282 22001478282 22001478282 22001478282 22001478282
[44,]  5001772265  5001772265  5001772265  5001772265  5001772265  5001772265
[45,] 11001679067 11001679067 11001679067 11001679067 11001679067 11001679067
[46,] 17001582342 17001582342 17001582342 17001582342 17001582342 17001582342
[47,] 19001574116 19001574116 19001574116 19001574116 19001574116 19001574116
[48,]  7001245393  7001245393  7001245393  7001245393  7001245393  7001245393
[49,]  1002073182  1002073182  1002073182  1002073182  1002073182  1002073182
[50,]  1002536319  1002536319  1002536319  1002536319  1002536319  1002536319
[51,]     1978379     1978379     1978379     1978379     1978379     1978379
[52,] 10001774925 10001774925 10001774925 10001774925 10001774925 10001774925
[53,] 13002153736 13002153736 13002153736 13002153736 13002153736 13002153736
[54,]  6001878723  6001878723  6001878723  6001878723  6001878723  6001878723
[55,]  9002423915  9002423915  9002423915  9002423915  9002423915  9002423915
[56,]  2001923778  2001923778  2001923778  2001923778  2001923778  2001923778
            [,19]       [,20]       [,21]       [,22]
 [1,]  4002079342  4002079342  4002079342  4002079342
 [2,]  2002031084  2002031084  2002031084  2002031084
 [3,]  9001820688  9001820688  9001820688  9001820688
 [4,] 34001679408 34001679408 34001679408 34001679408
 [5,] 11002232000 11002232000 11002232000 11002232000
 [6,]  7001872851  7001872851  7001872851  7001872851
 [7,] 17001835444 17001835444 17001835444 17001835444
 [8,] 24001762993 24001762993 24001762993 24001762993
 [9,]     2490933     2490933     2490933     2490933
[10,]     2147668     2147668     2147668     2147668
[11,]     2105298     2105298     2105298     2105298
[12,]     1864733     1864733     1864733     1864733
[13,]     2295745     2295745     2295745     2295745
[14,]     2334613     2334613     2334613     2334613
[15,]  7002054322  7002054322  7002054322  7002054322
[16,]     1912129     1912129     1912129     1912129
[17,]     2506797     2506797     2506797     2506797
[18,]     2667283     2667283     2667283     2667283
[19,]     2444337     2444337     2385567     2385567
[20,]     1297031     1297031     1297031     1297031
[21,] 11001971079 11001971079 11001971079 11001971079
[22,]     2259683     2259683     2259683     2259683
[23,] 12002017015 12002017015 12002017015 12002017015
[24,]  3001088451  3001088451  3001088451  3001088451
[25,]  7001551941  7001551941  7001551941  7001551941
[26,]  8001706941  8001706941  8001706941  8001706941
[27,]  3001195437  3001195437  3001195437  3001195437
[28,]     2498030     2498030     2498030     2498030
[29,]     2478148     2478148     2478148     2478148
[30,]     2554385     2554385     2554385     2554385
[31,]     2345474     2345474     2345474     2345474
[32,]     2750943     2750943     2750943     2750943
[33,]  1002597886  1002597886  1002597886  1002597886
[34,]     3077338     3077338     3077338     3077338
[35,]  2002755043  2002755043  2002755043  2002755043
[36,]  1002004219  1002004219  1002004219  1002004219
[37,]  1001967221  1001967221  1001967221  1001967221
[38,]     1797627     1797627     1797627     1797627
[39,] 19001612333 19001612333 19001612333 19001612333
[40,]  6001853616  6001853616  6001853616  6001853616
[41,]  4001882318  4001882318  4001882318  4001882318
[42,] 17001594059 17001594059 17001594059 17001594059
[43,] 22001478282 22001478282 22001478282 22001478282
[44,]  5001772265  5001772265  5001772265  5001772265
[45,] 11001679067 11001679067 11001679067 11001679067
[46,] 17001582342 17001582342 17001582342 17001582342
[47,] 19001574116 19001574116 19001574116 19001574116
[48,]  7001245393  7001245393  7001245393  7001245393
[49,]  1002073182  1002073182  1002073182  1002073182
[50,]  1002536319  1002536319  1002536319  1002536319
[51,]     1978379     1978379     1978379     1978379
[52,] 10001774925 10001774925 10001774925 10001774925
[53,] 13002153736 13002153736 13002153736 13002153736
[54,]  6001878723  6001878723  6001878723  6001878723
[55,]  9002423915  9002423915  9002423915  9002423915
```

```
[56,]  2001923778  2001923778  2001923778  2001923778

$no.candidates
[1] 22

$no.tasks
[1] 56

$no.subtasks
[1] 1

$no.experiments
[1] 1094

$no.alive
[1] 19

$alive
 [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE

$best
[1] 4

$mean.best
[1] 5948446599

$timestamp.start
[1] "Fri Apr 24 08:50:03 2009"

$timestamp.end
[1] "Fri Apr 24 08:56:33 2009"

$description.best
  label                command
4 1-1-8 runImpact.jar 10 10 80

$alive.inTime
[1] 22
```

B.2 ILS Permutation Tuning - R Output

```
Racing methods for the selection of the best
Copyright (C) 2003 Mauro Birattari
This software comes with ABSOLUTELY NO WARRANTY

Race name.......................Tuning of ILS Parameters on
                                                class-solomon2
Number of candidates........................................13
Number of available tasks...................................56
Max number of experiments..................................728
Statistical test.................................Friedman test
Tasks seen before discarding.................................6
Initialization function......................................ok
Parallel Virtual Machine.....................................no


                            Markers:
                               x No test is performed.
                               - The test is performed and
                                 some candidates are discarded.
                               = The test is performed but
                                 no candidate is discarded.


+-+-----------+-----------+-----------+-----------+-----------+
| |       Task|      Alive|       Best|  Mean best| Exp so far|
+-+-----------+-----------+-----------+-----------+-----------+
|x|          1|         13|         11|  1.747e+06|         13|
|x|          2|         13|          8|  5.162e+07|         26|
|x|          3|         13|          9|  6.819e+07|         39|
|x|          4|         13|          9|  1.264e+08|         52|
|x|          5|         13|          6|  1.415e+08|         65|
|=|          6|         13|          6|  1.181e+08|         78|
|=|          7|         13|          6|  1.015e+08|         91|
|-|          8|          7|          6|  1.015e+08|        104|
|=|          9|          7|          9|  9.029e+07|        111|
|=|         10|          7|          9|  8.138e+07|        118|
|=|         11|          7|          9|  7.407e+07|        125|
|=|         12|          7|          9|  6.797e+07|        132|
|=|         13|          7|          9|  6.282e+07|        139|
|=|         14|          7|          9|  5.841e+07|        146|
|=|         15|          7|          9|  5.458e+07|        153|
|=|         16|          7|          9|  5.122e+07|        160|
|-|         17|          3|          9|  4.826e+07|        167|
|-|         18|          2|          9|  4.563e+07|        170|
|=|         19|          2|          9|  4.328e+07|        172|
|=|         20|          2|          9|  4.115e+07|        174|
|=|         21|          2|          9|    4.4e+07|        176|
|=|         22|          2|          9|  4.203e+07|        178|
|=|         23|          2|          9|   4.46e+07|        180|
|=|         24|          2|          9|  4.694e+07|        182|
|=|         25|          2|          9|  4.509e+07|        184|
|=|         26|          2|          9|  4.338e+07|        186|
|=|         27|          2|          9|   4.18e+07|        188|
|=|         28|          2|          9|  4.036e+07|        190|
|=|         29|          2|          9|  3.902e+07|        192|
|=|         30|          2|          9|  3.775e+07|        194|
|=|         31|          2|          9|  3.657e+07|        196|
|=|         32|          2|          9|  3.547e+07|        198|
|=|         33|          2|          9|  3.443e+07|        200|
|=|         34|          2|          9|  3.345e+07|        202|
|=|         35|          2|          9|  3.252e+07|        204|
|=|         36|          2|          9|  3.167e+07|        206|
|=|         37|          2|          9|  3.085e+07|        208|
|=|         38|          2|          9|  3.008e+07|        210|
|=|         39|          2|          9|  2.933e+07|        212|
|=|         40|          2|          9|  2.864e+07|        214|
|=|         41|          2|          9|  3.041e+07|        216|
|=|         42|          2|          9|  3.209e+07|        218|
|=|         43|          2|          9|  3.369e+07|        220|
|=|         44|          2|          9|  3.523e+07|        222|
|=|         45|          2|          9|  3.447e+07|        224|
|=|         46|          2|          9|  4.027e+07|        226|
|=|         47|          2|          9|  3.943e+07|        228|
|=|         48|          2|          9|  4.488e+07|        230|
|=|         49|          2|          9|  4.603e+07|        232|
|=|         50|          2|          9|  4.513e+07|        234|
|=|         51|          2|          9|  4.426e+07|        236|
|=|         52|          2|          9|  4.343e+07|        238|
|=|         53|          2|          9|  4.452e+07|        240|
|=|         54|          2|          9|  4.371e+07|        242|
|=|         55|          2|          9|  4.293e+07|        244|
|=|         56|          2|          9|  4.219e+07|        246|
+-+-----------+-----------+-----------+-----------+-----------+

Selected candidate:           9 mean value:  4.219e+07

Description of the selected candidate:
   label              command
9 R25-E0 runILSRace.jar 25 -1


$precis
[1] "\nRacing methods for the selection of the best\nCopyright (C) 2003 Mauro Birattari\nThis software comes with ABSOLUTELY NO WARRA

$results
           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
 [1,] 101693139 101838462 201691201 101717873 201696242 101707540 101681751
 [2,]   1685588   1667377   1571879   1746904   1629472   1601070   1666714
 [3,] 101400601 101351287 101355637 101313931 101363605 101303096 101276698
 [4,] 501319678 301280322 301316323 701319824 401177083 401180132 301173274
 [5,] 101676552 101820849 201655302 101645680 201598169 101640097 201672492
 [6,]   1484003   1488833   1482303   1493530   1460930   1452241   1435121
 [7,]   1295539   1388317 101417296   1326622 101330480   1406890 101327410
 [8,] 201347590 201278857   1301481 201316602 201272816 101345220   1303807
 [9,]        NA   1281578   1294687        NA        NA   1331685   1312914
[10,]        NA   1361338   1317402        NA        NA   1269668   1226757
[11,]        NA   1004633   1033531        NA        NA   1062475   1010167
[12,]        NA    911982    892521        NA        NA    813681    874509
[13,]        NA   1067069   1071664        NA        NA   1080729   1080065
[14,]        NA   1130829   1027118        NA        NA   1033654   1086017
[15,]        NA    937216    947046        NA        NA    960596    936071
[16,]        NA    824941    853874        NA        NA    794912    806982
[17,]        NA   1036531    996755        NA        NA    973779    961861
[18,]        NA        NA        NA        NA        NA        NA   1052212
[19,]        NA        NA        NA        NA        NA        NA        NA
[20,]        NA        NA        NA        NA        NA        NA        NA
[21,]        NA        NA        NA        NA        NA        NA        NA
[22,]        NA        NA        NA        NA        NA        NA        NA
[23,]        NA        NA        NA        NA        NA        NA        NA
[24,]        NA        NA        NA        NA        NA        NA        NA
[25,]        NA        NA        NA        NA        NA        NA        NA
[26,]        NA        NA        NA        NA        NA        NA        NA
```

```
[27,]        NA        NA        NA        NA        NA        NA        NA
[28,]        NA        NA        NA        NA        NA        NA        NA
[29,]        NA        NA        NA        NA        NA        NA        NA
[30,]        NA        NA        NA        NA        NA        NA        NA
[31,]        NA        NA        NA        NA        NA        NA        NA
[32,]        NA        NA        NA        NA        NA        NA        NA
[33,]        NA        NA        NA        NA        NA        NA        NA
[34,]        NA        NA        NA        NA        NA        NA        NA
[35,]        NA        NA        NA        NA        NA        NA        NA
[36,]        NA        NA        NA        NA        NA        NA        NA
[37,]        NA        NA        NA        NA        NA        NA        NA
[38,]        NA        NA        NA        NA        NA        NA        NA
[39,]        NA        NA        NA        NA        NA        NA        NA
[40,]        NA        NA        NA        NA        NA        NA        NA
[41,]        NA        NA        NA        NA        NA        NA        NA
[42,]        NA        NA        NA        NA        NA        NA        NA
[43,]        NA        NA        NA        NA        NA        NA        NA
[44,]        NA        NA        NA        NA        NA        NA        NA
[45,]        NA        NA        NA        NA        NA        NA        NA
[46,]        NA        NA        NA        NA        NA        NA        NA
[47,]        NA        NA        NA        NA        NA        NA        NA
[48,]        NA        NA        NA        NA        NA        NA        NA
[49,]        NA        NA        NA        NA        NA        NA        NA
[50,]        NA        NA        NA        NA        NA        NA        NA
[51,]        NA        NA        NA        NA        NA        NA        NA
[52,]        NA        NA        NA        NA        NA        NA        NA
[53,]        NA        NA        NA        NA        NA        NA        NA
[54,]        NA        NA        NA        NA        NA        NA        NA
[55,]        NA        NA        NA        NA        NA        NA        NA
[56,]        NA        NA        NA        NA        NA        NA        NA
          [,8]      [,9]     [,10]     [,11]     [,12]     [,13]
 [1,] 101676031 101663403 101762558   1747145   1825355   1824957
 [2,]   1559973   1598734   1634375   1725025   1731383   1860763
 [3,] 101317876 101308897 101369086 101420453 101435126 101441879
 [4,] 601122435 301209203 701273506 501266652 501302212 501291747
 [5,] 101690869 201514987 101746247 101773418 101778023 201802248
 [6,]   1459034   1466745   1483049   1471734   1499565   1488446
 [7,] 101344130 101303440   1429876   1405121 101395049 201411637
 [8,] 101288559   1271688 201319265 101378569 101397981 201325029
 [9,]   1248128   1294075        NA   1329786        NA        NA
[10,]   1186819   1164008        NA   1300383        NA        NA
[11,]   1098172    981758        NA   1058403        NA        NA
[12,]    834931    849296        NA    934222        NA        NA
[13,]   1072057   1072271        NA   1067803        NA        NA
[14,]    969547   1054152        NA   1082875        NA        NA
[15,]    949024    922958        NA    939688        NA        NA
[16,]    797525    785240        NA    857446        NA        NA
[17,]    971011    961404        NA   1041948        NA        NA
[18,]   1028021    989458        NA        NA        NA        NA
[19,]    881885    864255        NA        NA        NA        NA
[20,]    776681    715073        NA        NA        NA        NA
[21,]   1023009 100998624        NA        NA        NA        NA
[22,]    742710    739598        NA        NA        NA        NA
[23,]    957395 100959860        NA        NA        NA        NA
[24,]    776677 100897859        NA        NA        NA        NA
[25,]    715221    686408        NA        NA        NA        NA
[26,]    588287    693175        NA        NA        NA        NA
[27,]    783246    721729        NA        NA        NA        NA
[28,]   1539244   1506010        NA        NA        NA        NA
[29,]   1372458   1343478        NA        NA        NA        NA
[30,]   1217679   1092757        NA        NA        NA        NA
[31,]    959068   1057721        NA        NA        NA        NA
[32,]   1478727   1455224        NA        NA        NA        NA
[33,]   1218698   1204990        NA        NA        NA        NA
[34,]   1124772   1083218        NA        NA        NA        NA
[35,]    899267    910576        NA        NA        NA        NA
[36,]   1710095   1672908        NA        NA        NA        NA
[37,]   1535131   1532458        NA        NA        NA        NA
[38,]   1309782   1330271        NA        NA        NA        NA
[39,]   1082780   1048158        NA        NA        NA        NA
[40,]   1505593   1516042        NA        NA        NA        NA
[41,] 101298337 101302552        NA        NA        NA        NA
[42,] 201068184 101082355        NA        NA        NA        NA
[43,]   1046700 101045036        NA        NA        NA        NA
[44,] 101306397 101211575        NA        NA        NA        NA
[45,] 101154904   1175018        NA        NA        NA        NA
[46,] 201176216 301105520        NA        NA        NA        NA
[47,]   1024789   1019795        NA        NA        NA        NA
[48,] 401063940 300975790        NA        NA        NA        NA
[49,]   1074125 100936716        NA        NA        NA        NA
[50,]   1020169   1133006        NA        NA        NA        NA
[51,]   1076236    908872        NA        NA        NA        NA
[52,]    865171    957022        NA        NA        NA        NA
[53,] 100899576 101133591        NA        NA        NA        NA
```

```
[54,]   1096625    987735        NA        NA        NA        NA
[55,]    990179    930371        NA        NA        NA        NA
[56,]    881755   1044550        NA        NA        NA        NA

$no.candidates
[1] 13

$no.tasks
[1] 56

$no.subtasks
[1] 1

$no.experiments
[1] 246

$no.alive
[1] 2

$alive
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE
[13] FALSE

$best
[1] 9

$mean.best
[1] 42185565

$timestamp.start
[1] "Mon Apr 27 21:44:26 2009"

$timestamp.end
[1] "Mon Apr 27 22:28:41 2009"

$description.best
   label             command
9 R25-E0 runILSRace.jar 25 -1

$alive.inTime
[1] 13
```

B.3 LO Pool Size Tuning - R Output

```
R version 2.6.2 (2008-02-08)
Copyright (C) 2008 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type ’license()’ or ’licence()’ for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type ’contributors()’ for more information and
’citation()’ on how to cite R or R packages in publications.

Type ’demo()’ for some demos, ’help()’ for on-line help, or
’help.start()’ for an HTML browser interface to help.
Type ’q()’ to quit R.

> library(race)
> source("race.STC.R")
> launch()

Racing methods for the selection of the best
Copyright (C) 2003 Mauro Birattari
This software comes with ABSOLUTELY NO WARRANTY

Race name..........................Tuning of LO pool size on
                                                class-solomon2
Number of candidates.........................................3
Number of available tasks...................................20
Max number of experiments...................................60
Statistical test.................................Friedman test
Tasks seen before discarding.................................6
Initialization function.....................................ok
Parallel Virtual Machine....................................no
```

```
                            Markers:
                               x No test is performed.
                               - The test is performed and
                                 some candidates are discarded.
                               = The test is performed but
                                 no candidate is discarded.


+-+-----------+-----------+-----------+-----------+-----------+
| |       Task|      Alive|       Best|  Mean best| Exp so far|
+-+-----------+-----------+-----------+-----------+-----------+
|x|          1|          3|          1|  9.002e+09|          3|
|x|          2|          3|          1|    1.4e+10|          6|
|x|          3|          3|          3|  1.767e+10|          9|
|x|          4|          3|          2|   2.25e+10|         12|
|x|          5|          3|          2|    2.2e+10|         15|
|=|          6|          3|          2|  2.117e+10|         18|
|=|          7|          3|          2|  2.557e+10|         21|
|=|          8|          3|          2|    3.2e+10|         24|
|=|          9|          3|          2|  2.956e+10|         27|
|=|         10|          3|          2|   2.76e+10|         30|
|=|         11|          3|          2|  2.655e+10|         33|
|=|         12|          3|          2|  2.567e+10|         36|
|=|         13|          3|          2|  2.477e+10|         39|
|=|         14|          3|          3|  2.193e+10|         42|
|=|         15|          3|          3|   2.14e+10|         45|
|=|         16|          3|          3|  2.169e+10|         48|
|=|         17|          3|          2|   2.23e+10|         51|
|=|         18|          3|          2|  2.178e+10|         54|
|=|         19|          3|          2|  2.253e+10|         57|
|=|         20|          3|          2|  2.155e+10|         60|
+-+-----------+-----------+-----------+-----------+-----------+

Selected candidate:           2 mean value:  2.155e+10

Description of the selected candidate:
  label              command
2 size5 runRacePoolsize.jar 5


$precis
[1] "\nRacing methods for the selection of the best\nCopyright (C) 2003 Mauro Birattari\nThis software comes with ABSOLUTELY NO WARRANTY\n\nRace name

$results
             [,1]        [,2]        [,3]
 [1,]  9001821587 13001892503 11001845705
 [2,] 19001545162 15001513106 18001479017
 [3,] 26001296132 26001229844 24001249406
 [4,] 41000928166 36001024548 43000827531
 [5,] 20001596653 20001576996 24001592205
 [6,] 25001410238 17001450493 18001321364
 [7,] 32000974927 52000813715 58000774086
 [8,] 45000906853 77000414322 40001028753
 [9,] 12001573456 10001560985 12001533312
[10,] 16001241266 10001428129 10001346343
[11,] 12001111605 16001196459 11001188390
[12,] 16000957035 16001069085 17001013617
[13,] 13001443954 14001389671 13001352503
[14,] 18001183610  9001371867  8001182919
[15,] 19000999637 18000984559 14000997101
[16,] 17000926348 18000992283 26000919084
[17,] 13001488634 12001249063 19001210413
[18,] 25001059907 13001398950 29001027997
[19,] 25000954086 36000924927  8001170930
[20,]  3000647998  3000701566  4000733889

$no.candidates
[1] 3

$no.tasks
[1] 20

$no.subtasks
[1] 1

$no.experiments
[1] 60

$no.alive
[1] 3

$alive
[1] TRUE TRUE TRUE
```

```
$best
[1] 2

$mean.best
[1] 21551209154

$timestamp.start
[1] "Tue Apr 28 09:46:08 2009"

$timestamp.end
[1] "Tue Apr 28 20:56:15 2009"

$description.best
  label               command
2 size5 runRacePoolsize.jar 5

$alive.inTime
[1] 3
```