Internal Report 2013-04

Universiteit Leiden
Computer Science

Sequential Parameter Tuning of Algorithms for the Vehicle Routing Problem

Name: Alexandru Florian Gaiu
Student-no: 1053396

Date: 13/02/2013

1st supervisor: Dr. M.T.M. Emmerich
2nd supervisor: Prof. Dr. T.H.W. Bäck

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The Vehicle Routing Problem is an extensively studied, NP-hard problem with numerous real-world applications, due to its super-polynomial time complexity exact approaches are infeasible, and so over the years metaheuristics have been used, such as the Genetic Algorithm(GA) and the Multiple Ant Colony Systems(MACS).

This paper extends the research on the Hybrid GA of Wink et al. and MACS-DVRPTW of van Veen et al. These approaches require a relatively large number of parameters, which in turn determines their effectiveness. In the past years, methods have arisen capable of automatically determining good parameter configurations, one of them being the Sequential Parameter Optimization Toolbox, which will also be discussed in this paper. In addition to solving all known problems optimally or within 2% of the optimum, SPOT aids in getting a better understanding of the parameters used by each algorithm.

Contents

Chapter 1

Introduction

The Vehicle Routing Problem (VRP) is one of the most studied combinatorial optimization problem, which is concerned with the optimal design of routes to be used by a fleet of vehicles to serve a set of geographically dispersed customers.

In the past years stronger and stronger algorithms emerged, trying to solve the VRP, some of which were successful in determining better solutions. Even so, one rule that holds true for all algorithms is the large number of parameters that influences their performance. With this in mind, this paper will focus in determining the efficiency and challenges of using an open-source parameter tuning mechanism, SPOT.

In order to evaluate SPOT's performance in tuning algorithms for the VRP, we shall test its robustness on two state of the art algorithms: the Hybrid-GA implemented by Wink et al. [43] and the MACS-DVRPTW of van Veen et al. [40].

The remainder of this work is structured as follows. First Chapter 2 will introduce Parameter Tuning. Then Chapter 3 offers an introduction to Vehicle Routing Problem. Following Chapter 4 presents the Meta-GA. Chapter 5 presents the second algorithm the MACS-DVRPTW. In Chapter 6 the tuning approach, SPOT will be described. The Experimental Setup is given in Chapter 7. Chapter 8 presents the results. Chapter 9 concludes this thesis.

Chapter 2

Parameter Tuning

Even though, methods for automatic parameter setting have emerged for more than 30 years, till recently most of the parameter setting was conducted manually. The change came as a result of the increased demand for a fast estimation of good parameter values [36].

Landgraaf et al. [15], state that there are two different methods in setting parameters values, and that is *parameter tuning*, which takes place before the beginning of the algorithm, and *parameter control*, that takes place during the run of the algorithm. Due to the nature of the algorithms to be tuned, in this paper we will focus our attention on parameter tuning.

Although parameter tuning was chosen as the approach for parameter setting, it has a drawback regardless of the method used (e.g. Meta-GA) in determining feasible parameter values, this is due to its inability to efficiently cope with the dynamic and adaptive nature of the metaheuristic algorithms. And so, the use of fixed parameters that do not change their values during the run of the algorithm may lead to lower performance.

Furthermore, it is considered, that different parameter values can be better at different stages of the evolutionary process[16]. For example, larger mutation steps can have a positive impact on early generations, by aiding the exploration of the search space and at the opposite end, small mutation steps might be more efficient in later generations. A solution to this problem would be to replace the parameters with functions that change their values over time.

It is widely accepted that the parameter values of all metaheuristic algorithms are of utmost importance as they have a significant impact on the algorithm's performance. However, finding feasible parameter values has several aspects, which makes it difficult in general:

- Parameter settings depend on each problem instance, such that a good configuration for one dataset may not lead to good results for another.
- Metaheuristics employ interrelated variables, whose values depend on each other. And so, tuning the parameters independently is not considered a feasible solution.
- In most cases the search space is extremely large due to several facts: primarily, an algorithm depends on many parameters, which have a wide region of interest, and at times they are sensitive to change.
- There is little knowledge regarding the influence of different parameters on the final result and the use of estimations offered by various theories are not always accurate.

When examining parameter tuning, Smit et al. [33] denoted the existence of three different layers. The first and also the lowest layer is the *application layer* and it corresponds to the problem, in our case the vehicle routing problem (VRP). The second, is the *algorithm layer*, representing the algorithm that aims to find optimal solutions for the VRPs. The top layer, *tuning layer* is the tuning method, which tries to find good parameter settings for the algorithm layer. The entire scheme can be seen as two different optimization problems, to which we can refer to as *problem solving* and *parameter tuning*. A schematic overview of the parameter tuning hierarchy is available in Figure 2.0.

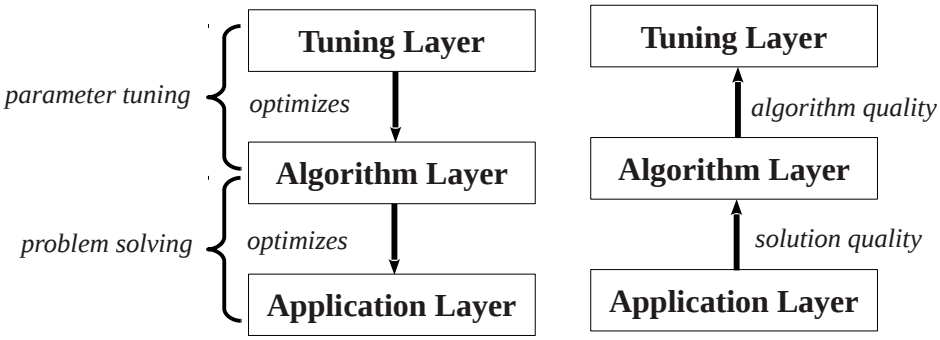

Figure 2.0 The three layer experimental setup. The left column represents the *control flow*, and the right column the *information flow.*

Parameter tuning [17] is comprised of two main tasks, the first one is determining the parameter sets that return the highest possible performance. This task can be seen as *exploitation* of knowledge regarding the parameter values. The second task is characterized by the accumulation of information on robustness, which implies *exploration* of the parameter space.

Every parameter tuning method is defined by a different balance between exploitation and exploration, and focuses on different types of robustness. This in turn leads to the differentiation of four main approaches. One that is mainly focused in achieving the best parameter, sets e.g. Meta-EAs, the second is focused in providing information e.g. sampling. And the other two approaches are a combination of both, for example screening and model-based approaches.

As our aim is to tune algorithms that solve the VRP, Golden et al. [19] state that the capabilities of metaheuristics are greater than that of classical heuristics. More precisely, the focus will be on population based metaheuristics, that make use of multiple solutions to search for an optimal result. Examples of such population based metaheuristics, are Swarm Intelligence, that mimic the collective behaviour of decentralized, self-organized systems (e.g. ants). Another broad category of population based metaheuristics are the Evolutionary Algorithms, which make use of nature inspired procedures such as mutation, reproduction and selection, to iteratively evolve solutions.

In the following two subsections we will focus our attention to the particulars of parameter tuning in the fields of Evolutionary Algorithms (EAs) and Swarm Intelligence (SI).

2.1 Parameter Tuning in Evolutionary Algorithms

EAs are a class of metaheuristic algorithms that are most commonly used to solve *NP*-hard optimisation problems. They can be described as a set of steps for identifying good results in a limited amount of time.

Even though there are many different evolutionary algorithms, we can describe EAs by considering a population of individuals within a given environment characterized by few resources, which determines the process of natural selection, also known as survival of the fittest. This in turn triggers a rise in the population's fitness values. This process can be artificially simulated, by considering an optimization function that has to be maximised. Then we can arbitrarily create a population of candidate solutions, to which we apply a fitness measure to determine the best candidates that will be selected to produce the next generation. Following, the selected individuals will suffer recombination and mutation.

The recombination operator is mainly applied to two or more individuals (parents) resulting in one or more individuals (offspring). The second operator, mutation is applied to only one individual resulting

in an altered version of himself. By applying recombination and mutation to the parents, this leads to the conception of a population of new individuals. Who will be evaluated based on their fitness values and depending on the selection mechanism/strategy will compete for a place in the new generation. The entire process can be repeated until an individual with sufficient quality is found or a previously set stopping criteria is reached.
A formal description of evolutionary algorithms is outlined in Algorithm 2.1.

Algorithm 2.1 Pseudocode generic EA

```
// Initialization
    t ← 0
    initpopulation ( P_t )
    evaluate ( P_t )

// Main loop
    while stop condition not met do
        P'_t ← selectparents ( P_t )
        recombine ( P'_t )
        mutate ( P'_t )
        evaluate ( P'_t )
        P_{t+1} ← select ( P'_t )
        t ← t + 1
    end while
```

One of the main challenges in the field of Evolutionary Computing (EC), is determining appropriate parameter values for EAs. As stated previously, researchers concur on the importance of good parameter values towards improving the performance of metaheuristic algorithms.
Nevertheless, the research is relatively limited, in terms of the studies conducted, on the effect of parameters on the EA's performance and on the methods of tuning these parameters. So far, the parameter values were largely selected by conventions (mutation success should be according to *1/5 rule*), and based on experimental comparisons conducted on small scale tests.

Now that we briefly described the evolutionary algorithms, we will focus on the state of the art parameter tuning methods. In order to decide upon a tuning method, we will examine the literature, and make a summary of the most commonly used and better reviewed alternatives. As it is beyond the scope of this thesis, we will not go in too much detail regarding the description or comparison of the methods.

Eiben et al. [17] conducted one of the most extensive research on parameter tuning for evolutionary algorithms, concluded in a comparison table (Appendix 1) of the previously identified four different parameter tuning approaches. The results in this paper and other papers concur that the Meta-GA[36] [15], REVAC[5][7][15][34][36], F-RACE[5][7] and SPOT[5][7][34][36] are one of the most efficient tuning methods for EAs. Meta-GA and REVAC are Meta-EAs, where the former is an enhanced approach. F-RACE is a screening method and SPOT is an iterative model-based method.

2.2 Parameter Tuning in Swarm Intelligence

A swarm is defined as a big population of homogeneous and simple agents that interact locally with themselves, and the surrounding environment, they lack central control and the knowledge of the global status of the swarm or its primary goal. Swarm-based algorithms have recently appeared as a group of nature inspired, population based algorithms that are able to determine fast and robust solutions to several *NP*-hard problems. Swarm Intelligence (SI) can be defined as a method used to mimic the collective behaviour of social swarms, such as ant colonies, honey bees, and bird flocks.

Even though, swarm individuals are fairly uncomplicated with limited abilities of their own, by successfully interacting with each other and using certain behavioural patterns they can achieve far greater tasks. This interaction amongst individuals is either direct or indirect. The former one takes place with the aid of visual or audio stimuli (e.g. the “dance” of honey bees). And on the other hand indirect interaction, takes place when individuals guide themselves based on the changes done to the environment by other individuals (e.g. pheromone deposit by ants). This paper will focus on the later.

When considering parameter tuning in SI, we have to research tuning approaches that work on the sub-field of interest, in our case Ant Colony Optimization (ACO), which will be discussed in greater detail in Chapter 5. Yuan et al. [44] mentions CALIBRA, F-RACE, SPOT and REVAC as tuning methods for ACOs.

2.3 Comparison and Selection of Parameter Tuning Approaches

In this subsection we will offer a short description, of the tuning methods that were mentioned as a feasible approach for both Evolutionary Algorithms and Swarm Intelligence. Then we will offer our decision upon the selected method that will be used in our analysis.

Relevance Estimation and Value Calibration (REVAC) [37] is an Estimation of Distribution Algorithm (EDA) that measures maximized entropy in the continuous domain. More precisely REVAC works by finding parameter vectors with high utility, collects the values of entropy for different utilities, and it creates a distribution for each parameter that indicates the expected utility of parameter values.

F-RACE [5] is inspired from racing algorithms, a method that iteratively evaluates a given set of candidate configurations on a stream of instances. And uses Friedman's rank test to eliminate unsuitable candidate solutions.

SPOT [8] is an implementation of the Sequential Parameter Optimization (SPO) framework, a heuristic which makes use of classical and statistical methods to improve the performance of search algorithms by building meta models based on the data collected from the exploration of the search space.

Literature research showed that all three methods proved to be efficient, however REVAC and SPOT, stand out. In terms of picking one over the other, we will consider the following aspects:

- Ability to tackle both numerical and categorical variables
- Ability to conduct multi-criteria optimization
- Flexibility towards tuning both EAs and ACOs
- Keep computational cost as low as possible
- Ease of implementation
- Offer detailed information about the parameter's influence on performance

In [17] Eiben et al., argues that there are few tuning methods capable of reducing tuning effort, such as SPOT and REVAC, where tuning effort is viewed as the product between the number of parameter sets to be tested and the number of algorithm iterations.

Two drawbacks of REVAC, are according to Smit et al. [32] its inability to handle parameter interactions (no joint distributions for multiple parameters) and that it cannot be used for tuning categorical parameters. Which is in contrast to Yuan et al. [44] who declared that REVAC can handle both numerical and categorical parameters. Yuan et al. also argues that SPOT is not able to handle categorical variables while Bartz et al [5] stated that it can, by encoding the variables as numerical values. In later papers[6][7][8] it is mentioned that SPOT uses factors to specify categorical variables.

In a comparative study [34] SPOT proved to be a high quality parameter tuner as it was able to determine good parameter values comparable to the ones found by Meta-EAs, plus it offers invaluable information through its resulting models.
Furthermore, SPOT includes methods to cope with stochastically disturbed results and it has been proven to be able to run on various metaheuristics including newer approaches such as algorithmic chemistries and particle swarm optimization. Also underlining SPOT flexibility [5], it can be used in fields such as: bioinformatics, water-resource management, mechanical engineering, biogas plant simulation, shipbuilding and quality control.

According to Bartz et al. [9], the main differences between SPOT and other tuning approaches are that, SPOT is able to maintain a relatively low computational cost while determining good parameter sets. A time constraint which in most cases rules out grid computing, local search methods and even Met-EAs. Furthermore, it offers detailed information that allows the user to learn, which is optional as the tuning process can run automatically. And probably the most significant difference is that SPOT can be applied in an algorithmic manner, it requires the specification of very few parameters and no major programming effort.

Based on the research conducted, we decided to use SPOT as a tuning method. Besides the previously mentioned factors, we considered SPOT also for the availability of its extensive documentation, the fact that it able to run in both Windows and Linux environments, and finally its ability to cope with algorithms implemented in relatively every programming language, in our case C and Visual C# (tests were done for JAVA also).

2.4 Research Objective

The main objective of this research is to determine the effectiveness of automated parameter tuning by testing SPOT on two different algorithms: Hybrid GA and MACS-DVRPTW, algorithms that try to solve the CVRP and DVRPTW respectively. This can be distributed among the following points:

Performance:

- *Is MSPOT able to match or improve the parameters found by the Meta-GA of Wink et al. [43]?*
- *Is SPOT able to match or improve manually tuned parameter sets for MACS-DVRPTW of van Veen [40]?*

Time:

- *Is MSPOT able to match or improve the time it took the Meta-GA to achieve good results?*

Robustness:

- *Do the parameter sets obtained by SPOT perform consistently?*

Information:

- *Is it possible to find a better understanding of the parameters used by each algorithm?*

To our knowledge, no research had been conducted in testing the usefulness of a tuning algorithm over two different metaheuristics coming from separate fields of natural computing, which are solving two different VRPs. And as such, we do believe that the results of this report will offer a better understanding of parameter tuning in general and be a motivation for further studies.

Chapter 3

Vehicle Routing Problem

According to Golden et al [20] the Vehicle Routing Problem (VRP) is a well known combinatorial optimization problem, which is concerned with the optimal design of routes to be used by a fleet of vehicles to serve a set of geographically dispersed customers.

From the moment it was first proposed more than 50 years ago, numerous papers have been dedicated to the precise and approximate solution of the many variants of this problem. Such as the Capacitated VRP (CVRP), in which a homogeneous fleet of vehicles is available and the only constraint is the capacity of the vehicles, or the VRP with Time Windows (VRPTW), where customers may be served within a given time interval and the schedule of the vehicle routes needs to be discover. Figure 3.0 depicts an example of the VRP with the optimal solution on the right.

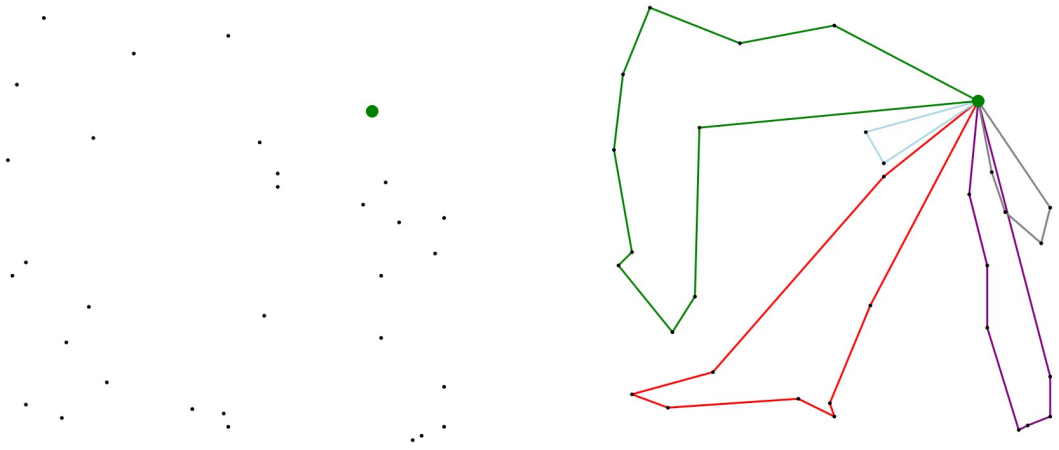

Figure 3.0: VRP example

[38] [26] In terms of complexity, the VRP is considered to be *NP*-hard and generalizes the Travelling Salesman Problem (TSP), which calls for the discovery of a minimum-cost circuit that visits all the vertices of G (a Hamiltonian circuit).

A problem which can be resolved using a polynomial-time algorithm (worst case complexity $O(n^k)$), is considered to be of complexity class *P*, which stands for polynomial time. A decidable decision problem whose solutions can be checked in polynomial time and its polynomial-time algorithm is unknown, falls under complexity class *NP* (non-deterministic polynomial time). Moreover, a problem is considered *NP*-hard if any problem in *NP* can be *reduced* to it in polynomial time.

Many approaches have been proposed for such problems, most of which simulate natural processes, e.g. biological evolution or ant colonies [24].

[27] As the VRP is a NP-hard problem, solving it optimality is not always possible within the limited computing time; in this situation, the solution should involve heuristic and meta-heuristic methods that can yield high-quality solutions in limited time.

3.1 Capacitated Vehicle Routing Problem

The Capacitated Vehicle Routing Problem (CVRP) is considered the basic VRP, and its mathematical model is defined on a graph $G(V,E)$ where:

- G is complete and symmetric.
- n is the number of customers.
- $v=\{v_0,v_1,...,v_n\}$ is the set of all vertices with depot v_0.The set of all customers corresponds to $V-v_0$.
- $E=\{(v_i,v_j)\in V:i\neq j\}$ is the set of all edges.
- $d(v_i):i\in\{1,..,n\}$ denotes the demand for customer i.
- $c_{ij}=\delta(v_i,v_j)=\sqrt{|v_{i_x}-v_{j_x}|^2+|v_{i_y}-v_{j_y}|^2}$ is the travel cost between vertices R_i, v_j.

  $c_{ij}=c_{ji}$ and $Cost(S)=\sum_{j=1}^{m} Cost(R_i)$ hold in CVRP.
- $R_i:i\in\{1,...,m\}=(v_0,v_{i_1},...,v_{i_{k(i)}},v_0)$ is a route for one vehicle.

Each route starts and ends at the depot.

- $S=\{R_1,...,R_m\}$ is the set containing all routes, forming the solution.
- $Cost(R_i)=\sum_{j=1}^{k(i)-1}(c_{i_j\,i_{j+1}})+c_{0i_1}+c_{i_{k(i)}0}$ is the travel cost for R_i.
- Every customer is visited exactly once by one vehicle:
  $\forall i(v_i\in V-v_0\Rightarrow\exists j:v_i\in R_j\wedge\nexists j\neq k:v_i\in R_j\wedge v_i\in R_k)$
- Total demand in routes does not exceed vehicle capacity K:

  $\forall i\in\{1,...,m\}:(\sum_{j=1}^{i_k} d(v_{i_j})\leqslant K)$

The goal is to minimize the total travel distance, thus the objective function is to be minimized.

$$Cost(S)=\sum_{j=1}^{m} Cost(R_i)$$

All vehicles have identical capacity, *Q*, and the number of vehicles is not determined a priori. The CVRP consists in determining a set of vehicle routes (a) starting and ending at the depot, and such that (b) each customer is visited exactly once, (c) the total demand of any vehicle route does not exceed Q, (d) the total cost of all routes is minimized.
As a solution for the CVRP, this paper will focus on the Meta-Genetic Algorithm of Wink et al. [43], further discussed in Chapter 4.

3.2 Vehicle Routing Problem with Time Windows

The Vehicle Routing Problem with Time Windows (VRPTW) it is based on the CVRP, and it has associated with each vertex $v_i\in V$ not only a demand $q_i\geq 0$ but also a service time $s_i\geq 0$ and a time window $[e_i,l_i]$. The depot has both the quantity (q_0) and the service time (s_0) equal to zero[29].

A viable solution for the VRPTW is defined as a set of routes that satisfy the following constraints: (a) each route starts and ends at the depot, (b) each customer is visited exactly once by a vehicle, (c) the total demand of customers in any route does not exceed Q and (d) the service of each customer is started between e_i and l_i.

As the analysis in this report is based on the work of van Veen et al. [40], it is important to state that the travelling distance and the travelling time of a route are equivalent, as in this case the travelling time does not include waiting time and service time.

3.3 Dynamic Vehicle Routing Problem with Time Windows

The Dynamic Vehicle Routing Problem with Time Windows (DVRPTW), is characterized [25] [30] by the fact that not all the necessary information for planning the routes is known to the user when the routing process starts and the information is bound to change after the initial routes have been constructed.

DVRPs' are also referred to as real time or online vehicle routing problems [22], and they can be dynamic in various ways [40], for instance dealing with changing travelling times, addition or deletion of nodes. The algorithm that we shall analyse in a later chapter focuses on the addition of nodes to the initial problem.

[22] Solutions to this type of problem range from linear programming to meta-heuristics, as it is beyond the scope of this thesis, we will only focus on van Veen et al. [40] algorithm: MACS-DVRPTW (Chapter 5).

3.4 Real world applications

[27] The VRP has drawn enormous interests from many researchers during the last decades because of its essential role in task sequencing, planning of distribution systems and logistics in many sectors such as parcel delivery, transportation of goods, snow ploughing. Furthermore, transportation is an active part in all stages of the production and distribution systems amounting up to 20% of the final cost of goods [38].

Based on the previous statement, it is fair to say that efficient routing solutions can save companies both time and money, and so, the market for efficient routing software is far from being saturated. A fact also proven by the extensive research into VRP, by a Norwegian research institute, SINTEF, which developed a software called Spider, coupled to real maps, thereby creating a usable solver for real world applications [21]. The software also runs problem instances from research literature, which enables performance comparisons.
Another example is Xtreme Route [51], a commercially available software which works on both research problem instances and actual maps, [43] and apparently holds the record for a large number of problem instances used in literature.

Chapter 4

Meta Genetic Algorithm

The focus of this chapter will be on the Meta-GA of Wink et al. [43]. The goal of their implementation was to develop an algorithm that could efficiently solve the CVRP, while at the same time offer the possibility to automatically determine good parameter sets.

The work of Wink et al. answered the following questions, which we shall cover in Section 4.3:

- *Is the Meta-GA able to match or improve manually tuned parameter sets?*
- *Is the Meta-GA able to match or improve the time it takes to manually tune a parameter set?*
- *Do the parameter sets obtained by the Meta-GA perform consistently?*

Before proceeding with their actual implementation we shall first offer a short introduction to GAs (Section 4.1) and the implemented Hybrid GA (Section 4.2).

This chapter will focus only on the key principles of the algorithm, in order to offer to the reader an understanding of the task. However, for a full description of the algorithm, please see Wink et al. [43].

4.1 Classical Genetic Algorithm

[18]Evolutionary Algorithms are metaheuristics inspired by Darwin's evolutionary theory, and they are used to determine solutions to combinatorial optimization problems through a repetitive process that aims to improve candidate solutions with the aid of genetically inspired operators.

Algorithm 4.1 Pseudocode Genetic Algorithm

```
// Initialization
    t ← 0
    initpopulation ( P_t )
    evaluate ( P_t )
// Main loop
    while stop condition not met do
        P'_t ← selectparents ( P_t )
        P''_t ← recombine ( P'_t , p_c )
        P'''_t ← mutate ( P''_t , p_m )
        evaluate P'''_t
        P_{t+1} ← select ( P'''_t )
        t ← t + 1
    end while
```

A subclass of Evolutionary Algorithms, the Genetic Algorithm (GA) is an adaptive strategy and a global optimization technique. It is inspired by population genetics including heredity and gene frequencies [10]. The GA uses operators like recombination, mutation and selection to improve candidate solutions and the evolution takes place at the population level. An outline is depicted in Algorithm 4.1.

In most cases the genetic operators are not capable to run on the natural representation of candidate solutions, the *phenotype space*, and so encoding/decoding is used to translate the phenotypes into *genotypes*. The translation is a simple conversion from integer to binary and the other way around.

4.1.1 Characteristics and Parameters

The initialization phase is defined by randomly generating individuals and evaluating them.
In the selection phase there is the option whether to include in the next generation, only the new offspring(λ) or to also add the originating parents(μ). The situation in which only the new offspring are included is called a *comma strategy* and the other a *plus strategy.*
Figure4.1.1 Depicts a situation in which the algorithm had stopped in local optimum. A situation that can be avoided by using a comma strategy that permits intermediate deterioration by forgetting highly fit individuals.

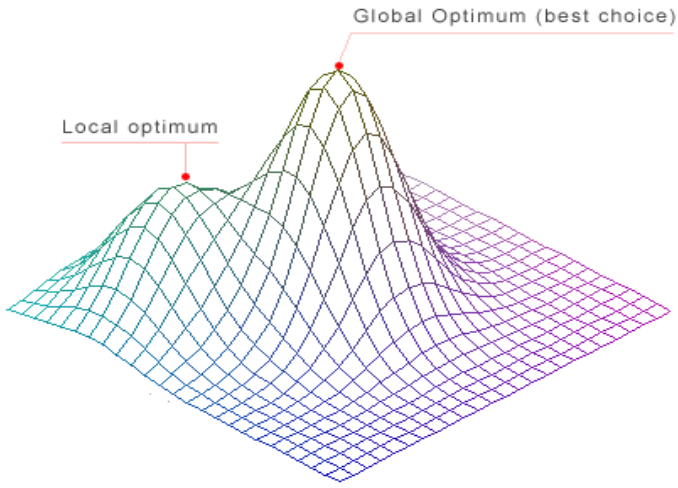

Figure 4.1.1 Local Optimum

The population size, also plays an important role in finding good solutions. A population that is too small may not be able to search enough of the solution space and λ should be chosen greater than μ to allow diversity.

Another parameter that influences the quality of solutions is the selective pressure, which emphasises on choosing the best individuals. A commonly used selection mechanism that offers increased control in selective pressure is tournament selection. It works by selecting the best individual from a randomly chosen pool of *K* individuals picked from the population.
In their algorithm Wink et al. [43], coupled tournament selection with ranking-based selection that offered even more control. The operator is outlined in Algorithm 4.1.1

Algorithm 4.1.1 Tournament selection with ranking

choose randomly *K* (the tournament size) individuals from the population
choose the best individual from the pool with probability *p*
choose the second best individual with probability $p * (1 - p)$
choose the third best individual with probability $p * (1 - p)^2$
and so on...

4.2 Hybrid Genetic Algorithm

In order to solve the CVRP, Wink et al. [43], introduced a Hybrid GA (HGA), different from the before mentioned Classical GA, the difference lies in the modified individual representations. The algorithm is outlined in Algorithm 4.2

Algorithm 4.2 Pseudocode Hybrid Genetic Algorithm.

```
// Initialization
    t ← 0
    initpopulation ( P_t )
// Main loop
    while stop condition not met do
        P'_t ← recombine ( P_t , p_c )
        P''_t ← mutate ( P'_t , p_m )
        P'''_t ← optimize ( P''_t , p_m )
        P_{t+1} ← select ( P'''_t )
        t ← t + 1
    end while
```

The most natural representation for a CVRP solution is a set of routes where each route starts and ends at the depot and contains the customers in the order they are visited. In VRP there are two common approaches in the permutation-based genotype design [43].

First one consists in encoding the phenotype by concatenating the customers in a single permutation and then, decoding is reached by assigning customers to vehicles. However, this approach is not capable to individually represent all possible phenotype, as after encoding/decoding, in some cases the result is different from the initial phenotype.

Wink et al. [43] used the second approach in terms of representation where genotype = phenotype. Meaning that the need for encoding/decoding functions was removed, however it implied the usage of specialized operators.
Such a system that uses other heuristics, domain knowledge or existing algorithms is referred to as hybrid evolutionary systems [2] and they are considered to be part of the class of 'hybrid meta-heuristics' [39].

In order to outline the distinctiveness of the HGA, following we shall shortly describe the implemented Local Search Heuristics (Section 4.2.1), Initialization (Section 4.2.2), Recombination (Section 4.2.3), Mutation (Section 4.2.4), Optimization (Section 4.2.5) and Selection (Section 4.2.6).

4.2.1 Local Search Heuristics

2-Opt [12], is a typical local search algorithm that basically removes intersections in routes with the goal of reducing the operational cost. It works by checking all pairs of non-adjacent edges for intersections. In the case that they do intersect 2-Opt, reorganizes the edges and so creating a route without intersections. The complexity is $O(n^2)$ since all pairs of edges are checked within each route.

Push Forward Insertion Heuristic (PFIH) [35], is an efficient method to create feasible solutions for any VRP by considering an infinite number of vehicles. PFIH works on lists of unrouted customers, whereupon creating a new route; firstly the most distant customer is inserted, followed by the most cost-efficient and feasible vertex whether it is a customer or a depot (in this is case a new route is created). PFIH is outlined in Algorithm 4.2.1

Algorithm 4.2.1 Tournament selection with ranking

```
i ← 1
while any unrouted customers do
    if Route i is empty then
      Find most distant customer and append it to Route i
    else
        vertex ← most cost-efficient insertion (feasible customers or depot)
        if vertex is a customer then
           Append customer to Route i
        else
            i ← i + 1
        end if
    end if
end while
```

4.2.2 Initialization

Kallel et al [23] stated that Initialization plays an important role in achieving good result using GAs, due to the fact that bad Initialization in the best case scenario it will increase the time to solution, whereas in the worst case scenario it could even prevent the convergence towards the global optimum.

Random initialization, a random CVRP solution is created using exhaustive routing by assigning customers to routes whilst capacity constraint is not broken.

Bearing initialization, used to steer the initial population towards a promising area in the search space. It is done by arranging the individuals from the initial solution in the direction of a 'butterfly' pattern. Starting at a certain bearing (i.e. 0^0, north) from the depot, the operator then scans clockwise for customers who are added exhaustively into the routes, without violating constraints [43]. In order to avoid the creation of duplicate individuals, different start bearings are used for each individual. The pseudocode is available in Algorithm 4.2.2

Algorithm 4.2.2 Bearing initialization

```
Require: 0 ≤ StartBearing < 360
Calculate bearing for each customer as seen from the depot
Order the customers by bearing as seen from the depot
Starting from StartBearing, exhaustively create routes
```

4.2.3 Recombination

Best Cost Route Crossover (BCRC), while checking for constraints this operator tries to simultaneously minimize the number of vehicles and cost. In this process two parents generate two offspring, by selecting one route from each parent whose customers are afterwards removed from the other parent.
The removed customers are sequentially re-inserted in the most cost-efficient location. In case of a stalemate (identical insertion costs in two or more positions), a random location is chosen from those positions. The operator is able to generate a new route when there are no viable insertion places, or if generating a new route results in the minimal increased travel distance.

Alvarenga Crossover (AX) [1], when creating the new offspring this operator tries to take as much complete routes as possible from both parents. Basically AX, generates an offspring resembling both parents equally, although this is largely dependent on the problem set. Moreover, if a route with many customers (pertaining to a large problem set with few vehicles) is inserted into the offspring, frequently no other route is possible for insertion any more. In this case, a large number of customers have to be routed using PFIH, which eventually will not yield good results.

4.2.4 Mutation

Merge routes, starts by unrouting the customers of a random number of randomly selected routes and reinserts them using the PFIH operator. The merge routes operator is most efficient in initial populations due to the fact that solutions display a relatively large number of overlapping routes in the early stages of the runs. However after the solutions have evolved, the operator is not capable of providing improvements in many occasions.

Adjacent reorder, this operator works by selecting a random customer and then identifying the nearest customer in another route. Followed by unrouting all customers in both routes and re-inserting them, using PFIH. And so two routes will be merged and they are most probably adjacent, which is precisely where the solution has a high likelihood of being improved.

4.2.5 Optimization

The currently studied GA has an additional step, optimization, where each individual is optimized using the 2-Opt heuristic.

4.2.6 Selection

Tournament selection is used, where $q>1$ individuals are chosen randomly from the population, out of that pool the best individual is being selected. Tournaments are being held for each individual of the new population. In this study, the tournament selection operator was extended by implementing a ranking-based selection within the pool. Furthermore, the operator is used only for selecting new individuals for the next generation.

4.3 Meta-GA

The performance of algorithms is directly proportional with the quality of the set of parameters used. In most cases, determining good configurations can be troublesome as most algorithms make use of many parameters, which in turn, results in an exponentially larger number of possible configurations.
A feasible approach in setting these parameters is to make use of “best practises” and a trial and error method, which is very time consuming.

Clune et al [11], proposes the use of a Meta-GA (GA within a GA) for the investigation of promising parameter settings with the goal of creating a self adaptive algorithm. This method was first used by Mercer and Sampson in 1978 and a more extensive investigation was conducted by Grefenstette 8 years later.

The Meta-GA works by optimizing parameter values, these values are encoded using a binary representation, and so for every parameter set, the Meta-GA sees a single individual. Whose, fitness value is evaluated by executing the lower level GA with the parameter set that belongs to the individual. As in the case of a classical GA, the individual with the highest fitness value will be selected for the next generation and when selected there is a chance it will undergo mutation or crossover.

The Meta-GA proposed by Wink et al., was implemented as a classical GA, that worked on the Hybrid GA described in Section 4.2. The implementation made use of classical operators such as 1-point crossover and bit-flip mutation together with a classical tournament selection. That left, population size, selection strategy and tournament size as the only parameters to be set for the Meta-GA.

In order to keep the bit-string of the genotype as short as possible, the parameters where encoded using *step sizes.* For example in the case of μ by using a step size of 2^n on a parameter range of [8 : 1024] resulted in an encoding of only 3 bits. Table 4.3 outlines the entire list of parameters, their range and step sizes.

Table 4.3 Meta-GA parameter coding

Parameter	Range	Step size	Bits
μ	8 : 1024	2^n	3
Selection Strategy	Comma or Plus selection	-	1
λ	2μ : 5μ	μ	2
Recombination Operator	BCRC or Alvarenga	-	1
Mutation Operator	Merge routes or Adjacent reorder	-	1
$P_{mutation}$	0.3 : 0.9	0.2	2
Tournament size	2 : 17	-	4
Initialization operator	Random or Bearing	-	1
Bit-string			15

The maximum number of generations was not considered for tuning based on the assumption that it should depend on the population size and it was set to 100,000/λ.

Using the set-up outlined in this section, the Meta-GA was able to match and at times even exceed manually tuned parameter sets, but only for problem instances up to 100 customers, as the running time for large instances was too long, reaching up to several days.

Another strength of Wink et al. Meta-GA is its ability to match and even improve the time it takes to manually tune a parameter set. Considering the situation that there is no prior knowledge on parameter settings, the algorithm can return in approximately 15 hours for a problem set of up to 80 customers a parameter configuration that yields a solution within 1% of the optimum.

Thirdly, the results of the Meta-GA are consistent as it has been proven, that running the same parameter set on the Hybrid GA will return good results.

Chapter 5

Ant Colony Optimization: MACS-DVRPTW

In this chapter we shall discuss the MACS-DVRPTW algorithm of van Veen et al. [40] proposed as a solution to the DVRPTW problem with different degrees of dynamism. Furthermore, this is considered to be the first ACO algorithm that solves the DVRPTW problem.

Due to the limited research on the topic, it is hard to compare the results of the algorithm on the DVRPTW with results from other papers. And so, we will compare our results only with those of van Veen et al. [40]

Before proceeding with the actual implementation of the algorithm we shall first offer a short introduction to Ant Colony Optimization (5.1) and MACS-VRPTW (5.2).

This chapter will focus only on the key principles of the algorithm, in order to offer the reader an understanding of the task, however for a full description of the algorithm please see van Veen et al. [40].

5.1 Ant Colony Optimization

Ant colony optimization is part of the larger field of swarm intelligence, a relatively new approach to problem solving, a system inspired by the social behaviour of insects and other animals. In particular, ants have inspired a number of methods and techniques among which are the ant algorithms[28]. In real life ants coordinate their activities through a form of indirect communication mediated by modifications of the environment, called stigmergy. For example, foraging ants deposit a chemical (pheromone) on the ground, which increases the probability that other ants will follow the same path.

One of the most successful example of an ant algorithm is the general purpose optimization technique known as ant colony optimization (ACO). ACO algorithms can be used to solve not only static but also dynamic combinatorial optimization problems[14]. The algorithm works by mirroring the self-organizing principles which determine the coordinated behaviour of foraging ants, in order to organise the cooperation of societies of artificial agents to solve computational problems.
A schematic representation of the foraging ants behaviour is illustrated in Figure 5.1

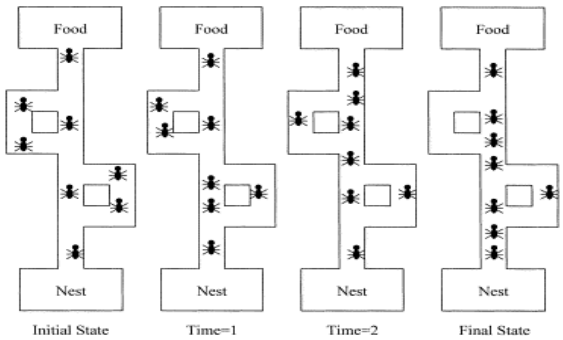

Figure 5.1 Schematic representation of ant behaviour.

An informal description of the ACO algorithm is given by Dorigo et al. [14], who describes it as the interaction of three main procedures: *ConstructAntsSolutions*, *UpdatePheromones*, and *DaemonActions*.

The first procedure, ConstructAntsSolutions, controls a population of ants that simultaneously and in an unsynchronised fashion, visits adjacent states of the problem by travelling through neighbour nodes of the problem's construction graph. They move based on a random local decision policy that makes use of pheromone trails and heuristic information. In this way, ants gradually create solutions to the problem in question. The moment an ant has created a solution, or during its development, the ant evaluates the (incomplete) solution that will afterwards be used by the UpdatePheromones mechanism to decide how much pheromone to deposit.

In the second process, UpdatePheromones, the pheromone trails are changed. Their value can be increased, as ants deposit pheromone on the components or connections they use, or lowered, as a result of pheromone evaporation. The deposit of new pheromone increases the probability that those components or connections that were used by a large number of ants and which produced a very good solution will be reused by future ants. On the other hand, pheromone evaporation applies a helpful form of forgetting, and as a result it avoids the premature convergence of the algorithm toward a suboptimal region, therefore enabling the exploration of new areas of the search space.

The DaemonActions mechanism is used to implement coordinated actions which cannot be performed by single ants. An example of such an action is the activation of a local optimization method, or the accumulation of global information that can be used to determine the usefulness of depositing additional pheromone to bias the search process from a non-local perspective. From a practical point of view, the daemon is capable to observe the solution found by every ant in the colony and select one or a few ants, mainly those that created the most promising solutions, and then allow them to deposit more pheromone on the components/connections they used.

An outline of the ACO pseudocode is available in Algorithm 5.1. The construct *ScheduleActivities* does not specify how the three processes described earlier, are scheduled or synchronized. More precisely it does not state whether they should be run in parallel, independent, or if some kind of synchronization among them is required. Therefore, the designer has the freedom to specify how these three procedures should interact, taking into consideration the characteristics of the problem at hand.

Algorithm 5.1 Pseudocode ACO algorithm.

```
procedure ACOMetaheuristic
    ScheduleActivities
        ConstructAntsSolutions
        UpdatePheromones
        DaemonActions % optional
    end-ScheduleActivities
end-procedure
```

5.1.1 Ant Colony System

[41] The basic ACO algorithm, the Ant System(AS) and its variants, have been successfully used on numerous optimization problems, such as the TSP, task scheduling, optimal path planning, load balancing and routing in telecommunication networks.

The two primary phases of the AS algorithm constitute the ants' *solution construction* and the *pheromone update*. [14] A good practice in initializing the pheromone trails is setting them to a somewhat higher value than the foreseen amount of pheromone deposited by the ants in one iteration; this value can be estimated with the following equation:

$\forall (i, j), \tau_{ij} = \tau_0 = m / L^{nn}$

With:

τ_o : starting pheromone value

L^{nn} : length of the nearest neighbour tour

m : number of ants

ij : represents the index corresponding to vertex (i, j)

The reasoning behind it, is that if the starting pheromone value τ_o is too low, then the search is rapidly biased by the first tours generated by the ants, which it will steer the exploration towards inferior zones of the search space. On the other hand, if the values are too high, then too many iterations will be wasted until pheromone evaporation reduces enough pheromone values, in order for the pheromone added by ants to be able to bias the search.

The solution construction, can be implemented in two different way, by using a parallel or sequential solution construction. In the former implementation, at every step all the ants in the colony move from their current position to the next one, while in the former implementation an ant creates a tour before the next one starts to build another one. In the AS case, the implementation options are considered equal [41] in the sense that they do not significantly alter the algorithm's behaviour. However, this does not hold true for other ACO algorithms such as Ant Colony System (ACS).

In terms of pheromone update, once all the ants have determined their solutions, the pheromone trails are updated. Firstly the pheromone values on all arcs are decreased by a constant factor, and then pheromone is added on the arcs the ants have crossed in their tours. In general, arcs that are used by larger numbers of ants and which are a part of shorter tours, will receive more pheromone and so will be more likely to appear in future iterations of the algorithm.

In contrast to Dorigo et al. [14], who states that ACS differs from AS in three main ways, van Ast et al. [41] mentions four differences, which will be now be summarized.

Firstly, the ACS algorithm implements a local pheromone update step, in order to avoid premature convergence to less favourable solutions, a step which occurs for each ant after each iteration within a trial, and it is defined by the following equation:

$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\,\tau_0$,

With:

ρ : pheromone evaporation, defined over the interval (0,1)

During the trial, the local pheromone update step, reduces the attractiveness of visited solution components in order to direct the ants towards less visited components.

Secondly, ACS makes uses of a global best update rule in terms of updating the global pheromone. Meaning that the solution with the highest fitness, found since the beginning of the algorithm is going to be used to update the pheromone variables at the end of the tour. This method used in ACO algorithms has proven to considerable speed up the convergence to the best solution.

Thirdly, the global pheromone update is executed just for the (i, j) pairs that are part of the global optimum solution. Meaning that not all pheromone levels are evaporated, just the ones that also receive a pheromone deposit.

Furthermore, the pheromone deposit is adjusted by ρ . As a result of the current and the previously mentioned two differences, the global pheromone update rule becomes:

$$\tau_{ij} = \begin{cases} (1-\rho)\,\tau_{ij} + \rho \sum_{k=1}^{m} \Delta\,\tau_{ij}^{k}, & \text{if } (i, j) \in T^{*} \\ \tau_{ij}, & \text{otherwise} \end{cases}$$

Where $\Delta \tau_{ij}^{k} = 1/L^{*}$

With:

T^{*}	: best tour found so far
L^{*}	: length of tour T^{*}
$\Delta \tau_{ij}^{k}$	: quantity of pheromone deposited by an ant k on vertex (i, j)
m	: number of ants

[13] Another notable difference between ACS and AS, consists in the decision rule used by ants during the construction process, called *pseudorandom proportional* rule. Which states that the probability for an ant to move from point *i* to point *j* depends on a randomly assigned variable *q*, from the interval [0, 1], and a parameter q_0 .

5.2 MACS-VRPTW

In Section 4.2 we introduced the Hybrid GA of Wink et al. [43], as a state of the art solution for the CVRP. In this section we will focus on van Veen's et al. [40] implementation of the Multiple Ant Colony System for the VRPTW (MACS-VRPTW), the algorithm is considered the most successful ACO for solving the VRPTW. As stated in Section 3.2, the VRPTW has two objective functions: to reduce the number of vehicles and the second, to reduce the total travel time. The antecedent has priority, as a solution with fewer vehicles and higher travel time is chosen over a solution with lower travel time but a higher number of vehicles. A schematic overview of the algorithm is outlined in Figure 5.2.

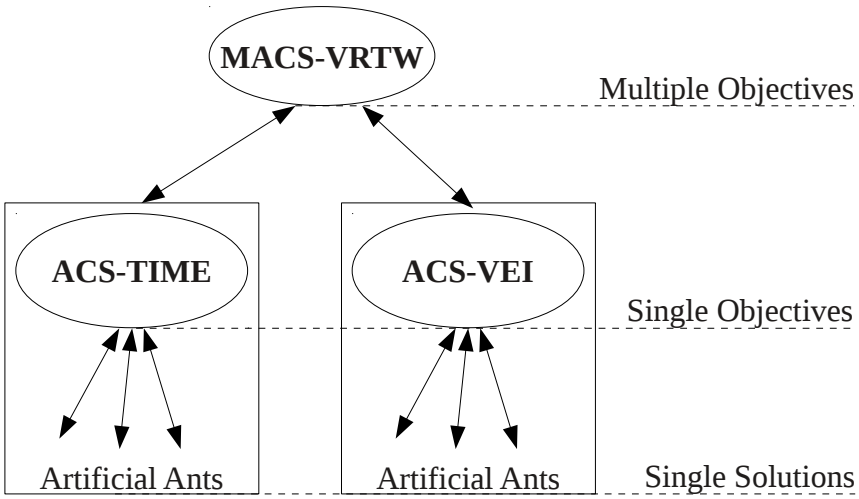

Figure 5.2: Overview of the MACS-VRPTW

As can be depicted from Figure 5.2, MACS-VRPTW optimizes a multiple objective function by coordinating two colonies, ACS-TIME and ACS-VEI, one for each objective. ACS-TIME, seeks to reduce the total travel time, while at the same time, colony ACS-VEI searches for a solution with fewer vehicles. Both colonies are based on the same solution construction procedure, similar to the one used by ACS. Furthermore, the two colonies run in parallel, while creating independent pheromone trails, nonetheless, they do collaborate by exchanging information through mutual pheromone updating.

ACS-VEI uses an array *IN*, in order to give increased priority to those nodes that had not been included in previous tours. At the same time it makes use of T^{VEI} to keep track of the best solution, which may not always integrate all nodes.

On the other hand ACS-TIME is not using a colony best solution, and it does not work with infeasible solutions, and it executes a local search procedure (Cross Exchange) to improve the solution.

Cross Exchange was implemented by van Veen et al., in such a way that MACS-VRPTW could handle hard time window constraints. Another adjusted method is the nearest neighbour heuristic, firstly the constraints on capacity and time windows are checked to avoid infeasible tours and secondly a limit on the maximum number of vehicles is passed to the function.

The algorithm begins with a feasible solution T^*, generated using the nearest neighbour heuristic. This solution is then optimized using the two colonies and when ACS-VEI finds a feasible solution with less vehicles, both colonies are restarted with the new reduced number of vehicles.

The pseudocode of the ACS-VEI and ACS-TIME can be found in Algorithms 5.2.1 and 5.2.2.

Algorithm 5.2.1 Pseudocode ACS-VEI(v)

Input: v is the maximum number of vehicles to be used
Given: n is the number of nodes

// Initialization
Pheromones are initialized to τ_0
IN initialized to 0
$T^{VEI} \leftarrow$ NearestNeigbour(v)

// Main loop
while stop condition not met **do**
for each ant k **do**
$T^k \leftarrow$ ConstructTour(k, IN)
for every nodes $i \notin T^k$ **do**
$IN_i = IN_i + 1$
end for
Update local pheromone on edges of T^k
$T^k \leftarrow$ InsertMissingNodes(k)
end for

Find ant l with most visited nodes
if nodes in T^l > nodes in T^{VEI} **then**
$T^{VEI} \leftarrow T^l$
IN $\leftarrow$ 0
if T^{VEI} feasible **then**
return T^{VEI} to MACS-VRTW
end if
end if

Update global pheromone with T^* using: $\tau_{ij} = (1-\rho)\,\tau_{ij} + \rho \sum_{k=1}^{m} \Delta\,\tau_{ij}^k, \forall (i,j) \in T^*$

Update global pheromone with T^{VEI} using: $\tau_{ij} = (1-\rho)\,\tau_{ij} + \rho \sum_{k=1}^{m} \Delta\,\tau_{ij}^k, \forall (i,j) \in T^{VEI}$

end while

Algorithm 5.2.2 Pseudocode for the ACS-TIME(v) algorithm.

Input: v is the maximum number of vehicles to be used
Given: n is the number of nodes

// Initialization
Pheromones are initialized to τ_0

// Main loop
while stop condition not met **do**
for each ant k **do**
$T^k \leftarrow$ ConstructTour(k, 0)
Update local pheromone for edges of T^k
$T^k \leftarrow$ InsertMissingNodes(k)
if T^k is a feasible tour **then**
$T^k \leftarrow$ LocalSearch(k)
end if
end for

Find feasible ant l with smallest tour length
if $\exists\, l : T^l$ is feasible **and** $L^l < L^*$ **then**
send T^l to MACS-VRTW
end if

Update global pheromone with T^* using: $\tau_{ij} = (1-\rho)\,\tau_{ij} + \rho \sum_{k=1}^{m} \Delta\,\tau_{ij}^k, \forall\,(i,j) \in T^*$

end while

5.3 MACS-DVRPTW

MACS-DVRPTW, is based on MACS-VRPTW adapted to solve the Dynamic VRPTW. As there is no prior knowledge of such an algorithm, the description will be solely based on van Veen et al.[40].The motivation to study the algorithm and implement an optimization method for it, is based not only on its novelty and the problem it solves, but also because of its different implementation. The algorithm was written in C on a Linux environment.

In order to solve the DVRPTW van Veen et al., implemented a method to simulate a form of dynamism, by introducing the concept of a working day of T_{wd} seconds. Which implies that a percentage of the total nodes will be excluded and given an *available time* at which they will become visible, the percentage represents the dynamism of the DVRPTW.

At the start of the algorithm a tentative tour is created with the available nodes, which will be updated during the simulation in order to constantly hold a solution to the current problem. The dynamic problem is split into n_{ts} static problems, which are solved sequentially, this is achieved by dividing the working day T_{wd} by n_{ts}. When a previously excluded node becomes available it is introduced into the tentative solution, the moment the following static problem starts.
The dynamism of problems makes it difficult to find and keep track of an optimal solution. In MACS-DVRPTW and MACS-VRPTW, the collective memory of the colony is reset each time the colony is restarted and so van Veen et al. made use of a *pheromone preservation*(γ) variable to store some part of the pheromone trails when the colonies are restarted. Now the pheromone levels are adjusted using the following formula: $\tau_{ij} = (1-\gamma)\,\tau_{ij} + \gamma\,\tau_0$.

MACS-DVRPTW has the same objectives as MACS-VRPTW, to reduce the total time and the number of vehicles. MACS-DVRPTW, starts by reading the problem instance and initializing the data structures, followed by creating a starting solution with the nodes available. Afterwards when a time slice starts, the newly available nodes are inserted using the InsertMissingNodes function.

Then the process continues by (re)starting, ACS-VEI and ACS-TIME. A pseudocode of MACS-DVRPTW is outlined in Algorithm 5.3.

Algorithm 5.3 Pseudocode MACS-DVRPTW

```
// Initialization
    T* ← NearestNeighbour(n)
    τ0 ← 1/nL^nn
    Initialize time t=0
    Initialize available nodes n
    Start counting CPU time t
    Activate ACS-VEI (v-1)
    Activate ACS-TIME (v)

// Main loop
  while t<T_wd do
        v ← get_vehicles( T* )
        while ACS-VEI and ACS-TIME active and time-step not over do
            Wait an improved solution T from ACS-VEI or ACS-TIME
            if get_vehicles(T) < v then
                kill ACS-TIME and ACS-VEI
            end if
            v ← get_vehicles(T)
            T* ← T
        end while

        if time-step over then
            if new part of T* is defined or new nodes are available then
                kill ACS-TIME and ACS-VEI
                Update available nodes n
                Insert new nodes into T*
                Commit to nodes in T*
            end if
        end if

        if colonies not active then
            Activate ACS-VEI (v-1)
            Activate ACS-TIME (v)
        end if

  return T*
  end while
```

Chapter 6

Sequential Parameter Optimization Toolbox

As we have already outlined in the previous chapters the algorithms to be tuned, in this chapter we will focus on the Sequential Parameter Optimization Toolbox (SPOT), as the method for tuning.

Before proceeding with the description of SPOT, we will first offer a description of the Sequential Parameter Optimization framework on which SPOT is based.

6.1 Sequential Parameter Optimization

Definition [4], Sequential Parameter Optimization (SPO) is a framework for tuning and understanding of algorithms by active experimentation. This approach makes use of methods both from computational statistics and exploratory data analysis, like *design of experiments*(DOE) and *design and analysis of computer experiments* (DACE). SPO can be interpreted as a search heuristic, that optimizes the performance of non–deterministic algorithms. The basic framework of SPO consists of a 12 step process outlined in Table 6.2.

Table 6.2 SPO framework [31].

1	Pre-experimental planning
2	Scientific claim
3	Statistical hypothesis
4	Specification of the - optimization problem - constraints - initialization method - termination method - algorithm main factors - initial experimental design - performance measure
5	Experimentation
6	Statistical modelling of data and prediction
7	Evaluation and visualization
8	Optimization
9	Termination: If the obtained solution is good enough, or the maximum number of iterations has been reached, go to step 11
10	Design update and go to step 5
11	Rejection/acceptance of the statistical hypothesis
12	Objective interpretation of the results from step 11

The DOE method relies on three steps: screening, modelling and optimization, where every step uses different experimental designs. The classic approach towards the design of experiments uses elements like linear regression models, however the shortcomings of this approach rely on the assumption that observation errors are independent. As these assumptions are only speculative SPO, uses a stochastic process model, DACE-Kriging, that helps predict unknown values and can be applied to interpolate results from extensive simulations.

[9] *Algorithm design* (D_A) and *problem design* (D_P) are the key elements of the SPO methodology. D_A defines ranges of values for the algorithm's variables, in other words a design point $x_a \in D_A$ represents a vector with the particular set of settings of the studied algorithm. On the other hand D_P specifies the variables related to the optimization problem (e.g. number of function evaluations). Together the two elements D_A and D_P form the *experimental design D.*

Determining good design points depends highly on the regression model used, however, identifying a feasible model depends on the design points, creating what it is known as the chicken and egg problem. In the area of parameter tuning for random search algorithms, research has proven the superiority of space design filling over classical factorial designs. One example of a space design filling model, is *Latin hypercube designs* (LHD), and it was integrated by SPO due to its ease of implementation and understanding. However, LHD has not been proven to have superiority over other models from its category, just over a number of simple random sampling designs.

6.2 Sequential Parameter Optimization Toolbox

[3][5] Sequential parameter optimization toolbox (SPOT) is one possible implementation of the SPO framework. SPOT, uses the available budget (e.g. number of function evaluations) in a sequential manner, such that it guides the search by building one or several meta models from the information gathered from the exploration of the search space.

The meta models are used to predict new design points and they are refined gradually, in order to improve knowledge about the search space. SPOT, copes with noise by improving confidence and it applies exploratory data analysis to learn from the tuning process. The tuning can take place both in an interactive or automated fashion.

The SPOT method is comprised of two phases, particularly the first is building the model and the second is the sequential improvement. A formal description can be viewed in Algorithm 6.2.

In the first phase a population of initial designs is determined from the algorithm's parameter space and then the algorithm is run *k* times for each design. The second phase is characterized by a loop of the following processes:

- A model based on the obtained data is build or updated.
- The predicted utility of the generated large number of design points is computed by sampling the model.
- The best design points are chosen and then the algorithm is run $k+1$ times for each of them.
- The new design points are added to the population and the loop is restarted unless the termination criteria is reached.

The variable *k* is incremented in each run, and it is used to identify the number of repetitions for every design. In order to obtain a comparable number of repetitions the best design points are rerun. Sequential approaches are considered to be more efficient than approaches that evaluate the information in one step, as they require a small number of function evaluations.

In order to determine the correlation between the algorithm's input and its output, SPOT uses a sequentially improved model, which has two main functions. The first, allows SPOT to find feasible parameters. And the second, determines the interaction between variables, which facilitates the understanding of how the algorithm works on a particular problem or how changes in this problem impacts the output. In terms of prediction models, regression and Kriging models or a combination of the two are the most commonly used, nevertheless, SPOT allows for the usage of large number of meta models.

Pseudocode SPOT

//phase 1, building the model
let *A* be the tuned algorithm
generate an initial population $X = \{\bar{x}^1, \ldots, \bar{x}^m\}$ of *m* parameter vectors
let $k = k_0$ be the initial number of tests for determining estimated utilities
for each $\bar{x} \in X$ **do**
    run *A* with $\bar{x}$ *k* times to determine the estimated utility *y* of $\bar{x}$
end for each

//phase 2, using and improving the model
while termination criterion not true **do**
    let $\bar{a}$ denote the parameter vector from *X* with best estimated utility
    let *k* the number of repeats already computed for $\bar{a}$
    build prediction model *f* based on X and $\{y^1, \ldots, y^{|X|}\}$
    generate a set X' of *l* new parameter vectors by random sampling
    for each bar $x \in X'$ **do**
        calculate $f(\bar{x})$ to determine the predicted utility $f(\bar{x})$ of $\bar{x}$
    end for each
    select set X'' of *d* parameter vectors from X' with best predicted utility ($d \ll l$)
    run A with $\bar{a}$ once and recalculate its estimated utility using all $k+1$ test results (improve confidence)
    let $k = k+1$
    run *A* *k* times with each $x \in X''$ to determine the estimated utility $\bar{x}$
    extend the population by $X = X \cup X''$
end while

SPOT uses simple text files as interfaces from the algorithm to the statistical tools. The files are divided in two sections, files that the user needs to provide and the files belonging to the SPOT output. A schematic overview of the files and their interaction with SPOT and the algorithm is available in Figure 6.3.

The input files that the user has to specify are as follow:
- *Algorithm design* (APD) files are used to specify the constant parameters used by the algorithm. However, APD files are not compulsory, and in our situation we are not going to make use of them as the parameters are already defined in the algorithms.
- *Region of interest* (ROI) files define the region over which the algorithm parameters are tuned. SPOT supports even categorical variables (e.g. the recombination operator), and they can be encoded as factors. In the Hybrid GA case we would have BCRC or AX recombination.
- *Configuration* files (CONF) are used to define SPOT specific parameters, such as the number of evaluations or the prediction model. Parameters without an assigned value will receive the default value. In the case that the file mode is disabled, this information will be stored in the *config* variable.

If the file mode is enabled SPOT creates the following output files:
- *Design file* (DES) specify D_A's. They are created automatically by SPOT and used by the optimization algorithm.
- *Result file* (RES), as the name, it stores the results of running the algorithm. These files are used in statistical evaluations and visualizations. SPOT uses RES files to generate prediction models.
- *Best file* (BST) stores the best results found in each sequential step, and they provide direct access to progress information.

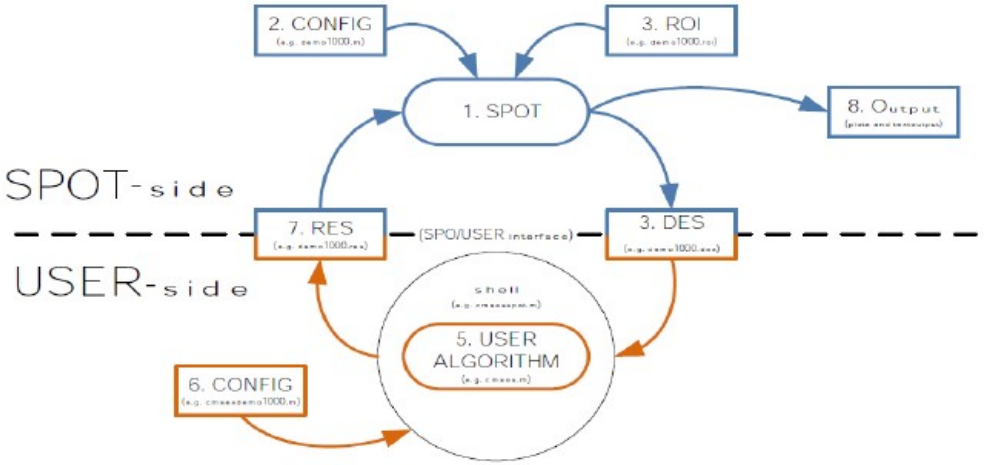

Figure 6.2 Overview of the SPOT files.

Following we will describe the 6 tasks that can be performed by SPOT.
1. *Initialize*. This is normally the first step during experimentation, and it generates an initial design. Before this step the user has to create the ROI and APD files and specify SPOT's parameters using a CONF file. When defining a project it is suggested to use the same base-name for CONF, ROI, and APD, however it is not compulsory, as one APD file can be used for different projects.
2. *Run*. In this task the optimization algorithm begins with configurations from the generated design, and additional information from the algorithms' D_P is also used. The step ends with results being written in the RES file.
3. *Sequential step*. Based on information from the RES file, a new design will be determined. This step is characterized by use of a prediction model. SPOT provides different prediction models, and we will use: *spotPredictForrester* for the Hybrid GA and *spotPredictRandomForest* for the MACS-DVRPTW. In the situations when only few algorithms runs are possible, and the focus is on efficiency it is possible to integrate user-defined models into SPOT.
4. *Report*. By using the information from the RES file, analysis can be generated. New report methods can easily be added as the information is stored in files. SPOT is equipped with scripts to conduct regression analysis and plots such as histograms, scatter plots, plots of the residuals, etc.
5. *Automatic mode*. As the name, the second and the third tasks, run and sequential, are performed automatically after an initialization for a user defined number of times.
6. *Meta mode*. This allows the tuning process to be repeated for a number of different configurations. For instance, tuning can be conducted for various starting points, several dimensions, or randomly chosen problem instances.

6.2.1 SPOT Multi-Criteria Optimization

In most cases industrial optimization problems, have more than one quality criteria. For example, besides the result itself, many optimization problems, have the computational time as the second criteria for performance. This is due to the fact that, time-consuming evaluations limit the optimization processes to a small number of evaluations.

In the past decade methods for multi-criteria optimization(MCO) emerged as a solution for problems with more than one quality criterion [45]. At the same time, it became necessary to use these MCO techniques together with optimization methods that require a small number of function evaluations.
Extended research has been conducted in combining MCO and surrogate model optimization, for example Voutchkov et al. [42] introduced a multi-criteria approach to sequentially improve surrogate models, tested on simple multi-criteria functions with a small number of function evaluations.

[46] Multi criteria SPOT (MSPOT) is similar to Voutchkov et al. approach, however it does not employ any form of expected improvement, or other forms of using the variance for exploration. MSPOT employs the surrogate models of the different objectives by making use of a multi-criteria optimization algorithm like SMS-EMOA or NSGA2; in our tests we used the former one.

This approach returns a population of promising points, where one or more of these points is/are selected in order to be used on the actual target function. This selection is based on non-dominated sorting and the individual hyper volume contribution. In order to avoid clustering of solutions in the objective space, the known points are tested again on the surrogate model.

Chapter 7

Experimental Setup and System Description

This section elaborates on the SPOT implementation details regarding the two algorithms meant to solve the CVRP and the DVRPTW respectively. The Hybrid GA was developed in Visual C# and the MACS-DVRPTW was implemented in C under a Linux environment.

7.1 Implementation Hybrid GA

As described in Chapter 2, the experimental setup consist of a three layer architecture, and Figure 7.1 outlines the hierarchy of our first implementation.

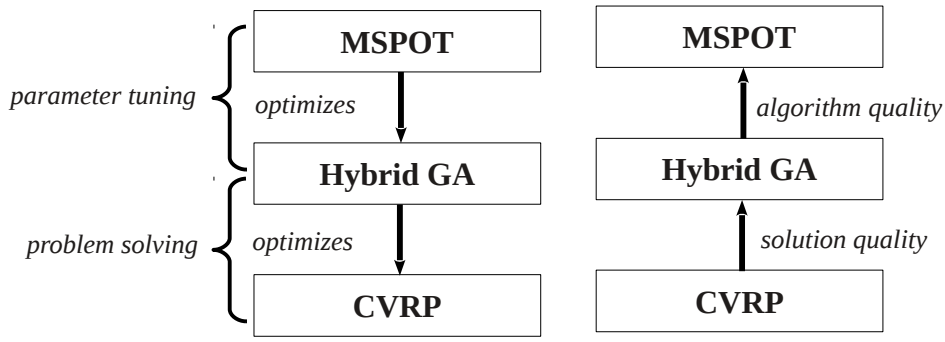

Figure 7.1 Experimental hierarchy MSPOT-HGA

7.1.1 Application Layer

On the application layer we have chosen, the 9 CVRP problems used by Wink et al. [43] in their tests of the Meta-GA. The size of the problems ranges from 60 to 80 clients, out of which 5 are clustered and 4 dispersed. A problem instance holds the coordinates and demands of all nodes. The distances between nodes are in Euclidean distance, however in order for the total travel distance to remain an integer, the distances are rounded to the nearest integer.

7.1.2 Algorithm Layer

On the algorithm level we used the HGA, described in Section 4.2. The actual implementation of the algorithm made use of multi-threading, allowing for a parallel iteration over λ individuals for each of the following steps: recombination, mutation, optimization, evaluation, and selection. This was one of the factors that greatly improved the HGA's performance.

In order to facilitate the interaction between SPOT and HGA, as previously depicted in Figure 6.2., firstly we created a new C# program(ConsoleProgram.cs) within the HGA, which is able to compile the algorithm with the parameter settings received from SPOT.

7.1.3 Tuning Layer

On the tuning layer we used a modified version of SPOT, more precisely MSPOT[46], a method for multi-criteria optimization. SPOT is implemented as an R package and to our knowledge, no prior research was conducted using this approach on an optimization algorithm implemented in Visual C#. This raised the problem of how to link R with C#, to this end we researched the following methods: R.NET [49], R(D)COM[48], SWIG [50]. However, these methods were not robust enough, and so another approach was required. After a lengthy trial and error process, the solution came in the form of two simple *callStrings* added to the SPOT's interface which set the connection to the algorithm.

```
# compiles the algorithm with the two required libraries
callString1 <- paste("csc ConsoleProgram.cs /r:Logic.dll /r:Data.dll")
callcs <-system(callString1, intern= TRUE)

# runs the algorithm with the currently tested parameter set
callString2 <- paste("ConsoleProgram", mu, lambda, ts, pm, ro, mo, io, ss )
y <-system(callString2, intern= TRUE)
```

In order to efficiently configure SPOT for a multi-criteria optimization process, it is important to choose a surrogate model based on the nature of the problem. As CVRP is a continuous problem with continuous parameters, it is considered that a Kriging model such as *spotPredictForrester* is the best option. Even so, spotPredictForrester, has its limitations when tuning categorical or boolean parameters.

An alternative approach, would be to tune the categorical parameters first, using a tree based model *spotPredictRandomForest*, and afterwards do a second run with the Kriging model to optimize the rest of the parameters. This approach did not yield better results, moreover our goal was not just to determine good parameter settings but also to get a better understanding of the parameters. Additionally we choose *nsga2*[45] as an optimization algorithm, and left Latin Hypercube Sampling, as the default method to optimize the model.

MSPOT was given a budget of 100 test evaluations, with an initial design size of 20. Each initial design point was set for 2 repeats and the sequential one for 3, totalling a number 40 individual configurations.

In our implementations we did not use a ROI file to define the region of interest, or any other file, as we implemented the necessary commands within the R script. The entire R script is available in Appendix 2.

As can be seen from Table 7.1.3, the HGA made use of four categorical parameters, theoretically they should have been encoded as factors, however, for consistency and to avoid any clash with the algorithm's implementation they were set as integers. Furthermore, the step sizes are handled by the console program, for example the value for $P_{mutation}$ is achieved using the following formula:
$P_{mutation} = 2\,pm + 1$, where $pm \in [1,4]$ is the value sent by MSPOT.

Table 7.1.3 Region of interest for the HGA

Variable Name	Lower	Upper	Type	Values HGA	Step size
μ	3	10	INT	8 : 1024	2^n
Selection Strategy	0	1	INT	Comma or Plus selection	-
λ	2	5	INT	$2\mu : 5\mu$	μ
Recombination Operator	0	1	INT	BCRC or Alvarenga	-
Mutation Operator	0	1	INT	Merge routes or Adjacent reorder	-
$P_{mutation}$	1	4	INT	0.3 : 0.9	0.2
Tournament size	2	17	INT	2 : 17	-
Initialization operator	0	1	INT	Random or Bearing	-

7.2 Implementation MACS-DVRPTW

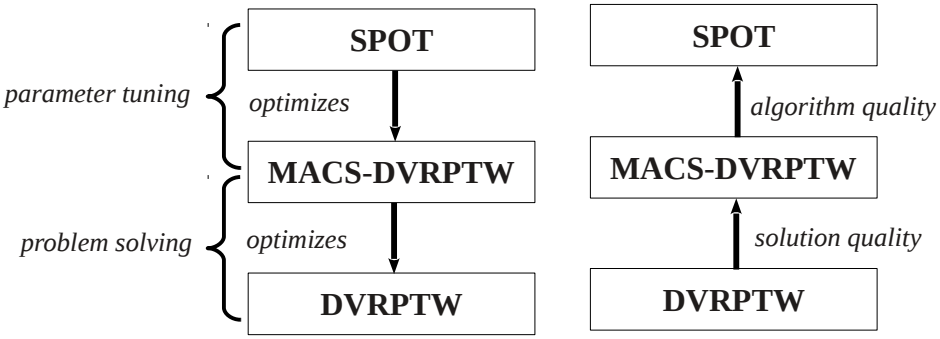

Figure 7.2 Experimental hierarchy SPOT-MACS-DVRPTW

7.2.1 Application Layer

We have tested SPOT, on 15 VRPTW from Solomon[35]. More precisely we choose the 9 R1 and 6 RC1 problems, where R has randomly placed nodes and RC has both randomly and clustered nodes. As mentioned in Section 5.3, the benchmark problems were modified with the goal to simulate different degrees of dynamism, by adding an availability time to each node.

7.2.2 Algorithm Layer

On the algorithm level we used MACS-DVRPTW, described in Section 5.3. The algorithm returns a detailed overview of the progress achieved by each of the two colonies(ACS-VEI and ACS-TIME), together with the distance and the number of vehicles of the best solution found.

Although informative, the output could not be interpreted by SPOT. Eliminating the extra text was straightforward, however as the solution itself is comprised of both the time and the number of vehicles. We had to convert the two values into one single number, achieved by multiplying the number of vehicles with 10^4 and added the *time* to this value, resulting in a number like this: 191652 where the first two digits represent the number of vehicles (19) and the rest the actual time (1652).

MACS-DVRPTW, was created as a collection of C programs and libraries that were compiled using a Makefile, which made it impossible for SPOT to assign parameter values directly to the program. To this end we created another C program (*Bridge.c*) that receives the parameters from SPOT and writes

them in a file(*Parameters.dat*), which the modified *program.c* can read and pass on to the algorithm in order to determine a solution.

7.2.3 Tuning Layer

To optimize MACS-DVRPTW, we made use of a single criteria optimization SPOT, whose configuration is very much alike that of MSPOT, and we will only underline the differences.
The connection with the application is done with the aid of four callStrings, grouped as two pairs. Each pair is preceded by a change of directory due to the fact that Bridge.c had to be placed in another folder as it clashed with Makefile, resulting in an error. The entire R script is available in Appendix 3.

```
setwd("/home/chill/Ma/Bridge/")
callString1 <- paste("gcc -o write Bridge.c")
call1 <-system(callString1, intern= TRUE)

callString2 <- paste("./write", iNumAnts, iRho, iAlpha, iBeta, iQZero, iPheromonePreservation)
call2 <-system(callString2, intern= TRUE)


setwd("/home/chill/Ma/")
callString3 <- paste("make")
callcs <-system(callString3, intern= TRUE)

callString4 <- paste("./main")
y <-system(callString4, intern= TRUE)

#callstring1 and callstring2, compiles the Bridge.c program and then writes the current parameter set into the Parameter.dat file, and callstring3 and callstring4 compiles and runs the algorithm
```

As MACS-DVRPTW was set to run for 100 seconds of CPU time, running a single criteria optimization was a better choice as the running time will always stay the same. Therefore, *spotPredictRandomForest*, became a more feasible approach for the surrogate model. Additionally we choose *cmaes*[6] to optimize on the surrogate model.

SPOT was given a budget of 200 test evaluations, with an initial design size of 30. Each initial design point was repeated 3 times and the sequential one for 4 times, totalling a number 57 individual configurations.

Table 7.2.3 Region of interest for the MACS-DVRPTW

Variable	Lower	Upper	Type
m	5	100	INT
ρ	0	1	FLOAT
α	1	5	INT
β	1	5	INT
q_0	0	1	FLOAT
γ	0	1	FLOAT

Where β is the influence of heuristic value on probability to be incorporated in tour

Chapter 8

Results

This chapter contains the results achieved by MSPOT and SPOT on the Hybrid GA and MACS-DVRPTW, respectively. Experimental runs are performed to get a better understanding of the parameters and how altering them affects the final result. Each section offers the final results of both implementations.

8.1 Results MSPOT-HGA

As can be seen from Table 8.1, MSPOT-HGA results are on average less than 0.6% over the optimum, results which are slightly worse than those of the Hybrid GA or the Meta-GA. However, the running time for MSPOT-HGA is less than half of that of the Meta-GA.

Table 8.1 Final results MSPOT-HGA

Problem		**Hybrid GA**		**Meta-GA**			**MSPOT-HGA**		
Instance	Optimum	Results	Over Optimum	Result	Time	Over Optimum	Result	Time	Over Optimum
A-n62-k8	1288	1300	0.93%	1288	10h	-	1300	4.1h	0.93%
A-n63-k9	1616	1627	0.68%	1616	10h	-	1616	3h	-
A-n64-k9	1401	1411	0.71%	1405	10h	0.29%	1414	4.3h	0.93%
A-n69-k9	1159	1159	-	1159	10h	-	1159	4.8h	-
A-n80-k10	1763	1766	0.17%	1764	15h	0.06%	1784	5.6h	1.19%
B-n63-k10	1496	1523	1.8%	1497	10h	0.07%	1517	3.8h	1.4%
B-n64-k9	861	861	-	861	10h	-	863	4.5h	0.23%
B-n68-k9	1272	1286	1.1%	1273	10h	0.08%	1275	6.6h	0.24%
B-n78-k10	1221	1221	-	1221	15h	-	1222	6.9h	0.08%

When running MSPOT for a multi-criteria optimization, we used *computational time* as the second criteria. The tests proved that such an approach is not really needed, as the results for the *computational time* never influenced the final solution due to its small variations between different configurations. We can conclude that different parameter settings do not influence the time it takes the algorithm to optimize the CVRP.

In their tests Wink et al. used an *AMD Phenom II X4 940, 3.0 GHz CPU*, a processor that outperforms an *Intel i3 330M, 2.13 GHZ CPU* which was used for the tests in this paper. According to [47] the AMD processor has a 3711 CPU Mark whereas the Intel has only a 1811 CPU Mark, although we can not state for certain that the former processor performs twice as good, it is fair to say that it has twice as many cores which aids with the parallel threading of the HGA implementation.

The difference in processing power might not influence the quality of the results too much but it will definitely allow MSPOT to finish its evaluations faster, a fact proven by the up 50% increase in the time it took us to optimize a problem instance using the HGA. Overall, we consider that using a more powerful processor will allow MSPOT to run a larger number of evaluations in the same amount of time, which in turn raises the probability of achieving better results.

8.2 Results SPOT-MACS-DVRPTW

As can be seen from Table 8.2, the parameter sets found with SPOT, perform on average with 2% worse than the best found results by MACS-DVRPTW. The running time for all problems was of 5.7 hours.

Table 8.2.1 Final results SPOT-MACS-DVRPTW

Problem	**MACS-DVRPTW**		**SPOT-MACS-DVRPTW**			
Instance	No Vehicles	Result	m/ ρ / α / β / q_0 / γ	No Vehicles	Result	Over Best Solution
r101	19	1650	23/ 0.1/ 3/ 2/ 0.8/ 0.6	19	1652	0.12%
r102	17	1486	25/ 0.6/ 4/ 5/ 0.4/ 0.5	17	1488	0.13%
r103	13	1292	27/ 0.1/ 4/ 2/ 0.1/ 0.2	13	1333	3.17%
r104	10	986	13/ 0.1/ 3/ 2/ 0.9/ 0.5	10	1008	2.23%
r105	14	1377	33/ 0.3/ 3/ 3/ 0.4/ 0.5	14	1403	1.9%
r106	12	1259	18/ 0.2/ 5/ 4/ 0.4/ 0.7	12	1291	2.54%
r107	10	1119	16/ 0.2/ 4/ 4/ 0.1/ 0.8	10	1150	2.77%
r108	9	974	7/ 0.1/ 3/ 4/ 0.8/ 0.7	10	967	N/A
r109	11	1211	18/ 0.1/ 2/ 4/ 0.6/ 0.1	11	1242	2.55
rc101	14	1696	35/ 0.0/ 3/ 2/ 0.6/ 0.2	14	1703	0.41%
rc102	13	1477	9/ 0.5/ 4/ 4/ 0.2/ 0.8	13	1484	0.05%
rc105	14	1540	5/ 0.3/ 2/ 3/ 0.6/ 0.9	13	1667	-
			7/ 0.5/ 2/ 3/ 0.6/ 0.9	14	1568	1.82%
rc106	12	1384	9/ 0.5/ 4/ 4/ 0.2/ 0.8	12	1396	0.87%
rc107	11	1232	14/ 0.3/ 3/ 4/ 0.4/ 0.7	11	1230	-0.16%
rc108	10	1139	9/ 0.5/ 4/ 4/ 0.2/ 0.8	10	1150	0.97%

According to van Veen et al. [40] the Cross Exchange local search can at times produce a tour with an empty vehicle, which happens for rc103 and rc104, making it impossible for SPOT to tune.

In their tests van Veen et al. also made use of a more powerful processor (*Intel i5, 3.2 GHz CPU)*, which also has a CPU Mark above 3000. In this situation we are convinced that the quality of the results is directly proportional with the speed of the processor, as the MACS-DVRPTW was set to run for 100 seconds of CPU time. By calculating the difference in processing speed, Intel i5 is capable to do up to 50% more calculations per second.

Besides finding good parameter sets, SPOT offers an overview of the parameters that had the greatest impact towards good results (Figure 8.2.1), and also their progress (Figure 8.2.2). By comparing the schematic outputs of different problems we can get a better understanding of which parameters have the greatest impact on the algorithm's performance. In the case of RC problems: β followed by γ or m have the greatest impact.

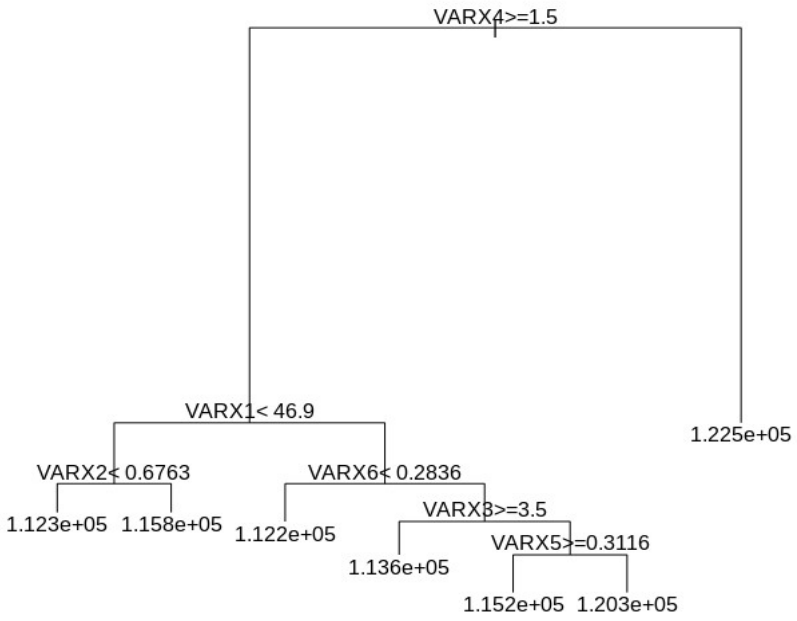

Figure 8.2.1 Final output of the RC107.0.0 problem instance. (VARX4 is the 4th variable in the sequence: *m*/ ρ / α / β / q_0 / γ)

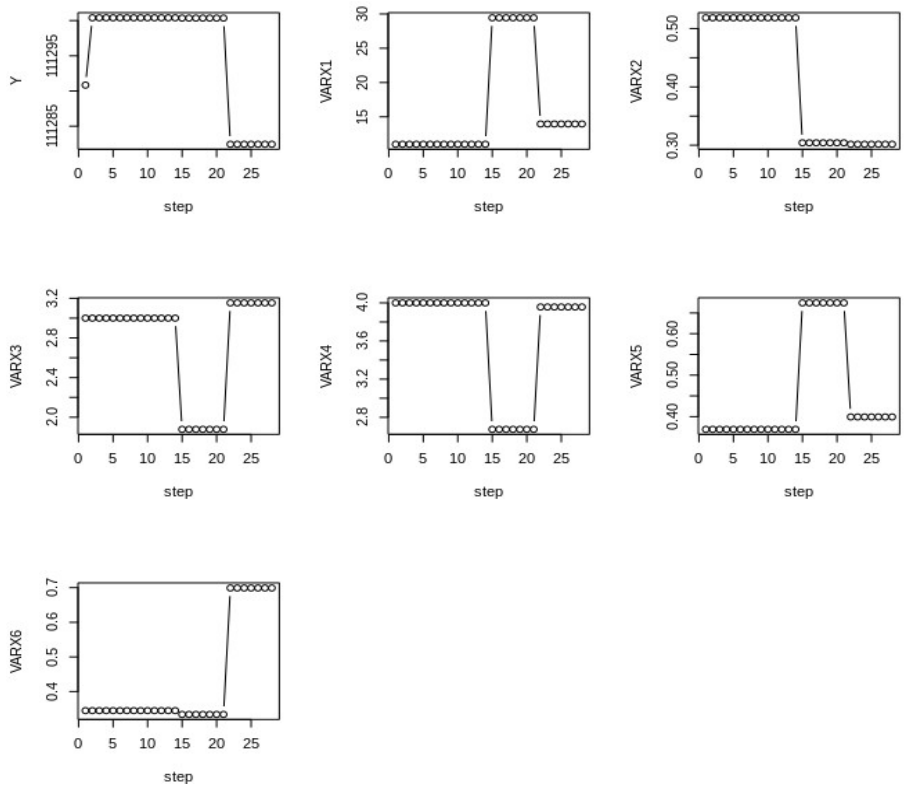

Figure 8.2.2 Progress Overview.

Chapter 9

Conclusion and Outlook

The focus of this thesis is to determine the effectiveness of automated parameter tuning by testing SPOT on two different algorithms: HGA and MACS-DVRPTW, algorithms that try to solve the CVRP and DVRPTW respectively. By answering the five questions defined in Section 2.4, we will outline the effectiveness of this approach.

Is MSPOT able to match or improve the parameters found by the Meta-GA of Wink et al.[43]?
No, it is not, as the results are on average less than 0.5% over the results achieved by the Meta-GA. However, as specified in Section 8.1, the processor used in Wink et al.'s tests is up to twice as powerful as the one used to test the implementation presented in this paper. With this in mind we consider that in order to accurately determine the gap in performance between the two approaches testing SPOT on an evenly performant computing system is imperative.

Is SPOT able to match or improve manually tuned parameter sets for MACS-DVRPTW of van Veen [40]?
The answer is also no, the results being on average 2% worse than the results achieved with the default parameters. We consider that this might be due to the same issue, the processor performance, in this situation being even more emphasised by the fact that the algorithm was set to run for 100 seconds CPU time. With much lower processor speed (up to 50%), the algorithm is stopped at an earlier stage in its optimization. In order to confirm our results we also suggest testing our implementation with a better system.

Is MSPOT able to match or improve the time it took the Meta-GA to achieve good results?
Yes, MSPOT managed to achieve high-quality results in only half the time of the Meta-GA, with only 200 evaluations instead of 310.

Do the parameter sets obtained by SPOT perform consistently?
Yes, they do, as tests proved that running the algorithms with the parameters found with SPOT returns good results.

Is it possible to find a better understanding of the parameters used by each algorithm?
We consider that by analysing and comparing the graphical representation generated by SPOT, we can determine which parameters have the greatest impact on the performance of the algorithm based on the type of problem used.

To summarize, we consider that SPOT, has a strong potential to determine good parameter settings for various algorithms regardless of the type of problem used. Although, our tests failed to prove SPOT's ability to achieve optimum parameters, it did however show that it is capable to achieve good settings in half the time of a Meta-GA approach.

In order to determine the concrete utility of SPOT, more tests need to be done on a more performant computer, or at least allowing it to run for the same duration as the Meta-GA.
In terms of configuration, using MSPOT on the CVRP was blunt, however using it with the second objective as the standard deviation of the vector of solutions it might determine better results[45]. For this purpose we do suggest another set of tests with this particular configuration on both algorithms.

Bibliography

[1] G.B. Alvarenga, G.R. Mateus, and G. de Tomic. *A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows.* Computers & Operations Research, Vol 34. 2007

[2] Th. Back, D.B. Fogel and Z. Michalewicz. *Handbook of evolutionary computation.* IOP Publishing Ltd. Bristol, UK. 1997

[3] Th. Bartz-Beielstein. *Sequential Parameter Optimization An Annotated Bibliography*. 2010

[4] Th. Bartz-Beielstein, M. Chiarandin, L. Paquete and M. Preuss. *Experimental methods for the analysis of optimization algorithms*. Heidelberg: Springer, p35. 2010.

[5] Th. Bartz-Beielstein, M. Chiarandin, L. Paquete and M. Preuss. *Experimental methods for the analysis of optimization algorithms*. Heidelberg: Springer, p287-363. 2010.

[6] Th. Bartz-Beielstein, M. Preuss, and M. Zaefferer. *Statistical Analysis of optimization Algorithms with R*. 2012

[7] Th. Bartz-Beielstein, *spot: An R Package For Automatic and Interactive Tuning of Optimization Algorithms by Sequential Parameter Optimization.* 2010

[8] Th. Bartz-Beielstein, O. Flasch, P. Koch, and W. Konen. *SPOT: A Toolbox for Interactive and Automatic Tuning in the R Environment.* In: Proceedings 20. Workshop Computational Intelligence. p264-273. 2010

[9] Th. Bartz-Beielstein, C. Lasarczyk, M. Preuss. *Sequential Parameter Optimization*. In: B. McKay, Proceedings 2005 Congress on Evolutionary Computation (CEC'05), Edinburgh, Scotland Bd. 1. Piscataway NJ : IEEE Press, S773-780. 2005.

[10] J. Brownlee. *Clever algorithms: nature-inspired programming recipes*. S.l.: Lulu Com. 2011.

[11] J. Clune, S. Goings, B. Punch, and E. Goodman. *Investigations in meta-GAs: panaceas or pipe dreams?* In GECCO Workshops, p235–241. 2005

[12] A. Croes. *A method for solving traveling salesman problems.* Operations Research, Vol 5, p791–812. 1958

[13] M. Dorigo, M. Birattari, and Th. Stützle. *Ant Colony Optimization Artificial Ants as a Computational Intelligence Technique*. IEEE Computational Intelligence Magazine. Free University Brussels, Belgium. 2006

[14] M. Dorigo and Th. Stützle. *Ant Colony Optimization*. Massachusetts Institute of Technology. 2004

[15] W.A. de Landgraaf, A.E. Eiben and V. Nannen. *Parameter Calibration Using Meta-Algorithms.* 2006

[16] A.E. Eiben, R. Hinterding, Z. Michalewicz. *Parameter control in evolutionary algorithms*. Evolutionary Computation, IEEE Transactions on, vol.3, no.2, p124-141. 1999

[17] A.E. Eiben and S.K. Smit. *Parameter Tuning for Configuring and Analyzing Evolutionary Algorithms.* Swarm and Evolutionary Computation. 2011.

[18] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. New York: Springer. 2003.

[19] B. Golden, E. Wasil, J. Kelly and I.-M. Chao. *The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results*. In: T. Crainic, G. Laporte(Eds). Fleet management and logistics. Boston: Kluwer; p33–56. 1998

[20] B. Golden, S. Raghavan, and E. Wasil. *The Vehicle Routing Problem: Latest Advances And New Challenges*. Springer Science+Business Media, LLC. 2008.

[21] G. Hasle. *Vehicle routing in practice*. Presentation at XVIII EWGLA, Naples, Italy. 2010

[22] P. Jaillet and M. R. Wagner. *Generalized online routing: New competitive ratios, resource augmentation, and asymptotic analyses.* Operations Research. 2008

[23] L. Kallel and M. Schoenauer. *Alternative Random Initialization in Genetic Algorithms*.

[24] H. Kwasnicka. *Efficiency Of Selected Meta-Heuristics Applied To The Tsp Problem: A Simulation Study.* Wroclaw University of Technology. 2002

[25] A. Larsen. *The Dynamic Vehicle Routing Problem.* IMM, DTU. 2000

[26] J.K. Lenstra and A.H.G. Rinnooy. *Complexity of vehicle routing and scheduling problems*. Networks Vol 11, p221–227, 1981

[27] C.Y. Liong, I. Wan Rosmanira, O. Khairuddin and M. Zirour. *Vehicle Routing Problem: Models And Solutions.* University Kebangsaan Malaysia. 2008

[28] V. Maniezzo, L.M. Gambardella and F. de Luigi. *Ant Colony Optimization.* Springer-Verlag, Berlin Heidelberg, p21. 2004.

[29] Y. Nagata. *Efficient Evolutionary Algorithm for the Vehicle Routing Problem with Time Windows: Edge Assembly Crossover for the VRPTW.* Japan Advanced Institute of Science and Technology. 2007.

[30] V. Pillac, M. Gendreau, C. Gueret, and A.L. Medaglia. *A review of dynamic vehicle routing problems*. European Journal of Operational Research. CIRRELT. 2011.

[31] M. Preuss and Th. Bartz-Beielstein. *Sequential Parameter Optimization Applied to Self-Adaptation for Binary-Coded Evolutionary Algorithms.* Berlin Heidelberg. Springer. 2007

[32] S.K. Smit and A.E. Eiben. *Using Entropy for Parameter Analysis of Evolutionary Algorithms.* In: Experimental methods for the analysis of optimization algorithms. Heidelberg: Springer, p298-301. 2010.

[33] S. K. Smit and A. E. Eiben. *Parameter tuning of evolutionary algorithms: Generalist vs. specialist.* Applications of Evolutionary Computation, vol. 6024 of Lecture Notes in Computer Science. Springer. 2010.

[34] S.K. Smit, A.E. Eiben, C*omparing parameter tuning methods for evolutionary algorithms.* In: Proceedings of the 2009 IEEE Congress on Evolutionary Computation, IEEE Press, Trondheim, p399-406. 2009.

[35] M.M. Solomon. *Algorithms for the vehicle routing and scheduling problems with time window constraints.* Operations Research, Vol 35, p254–265. 1987

[36] R. Stoean, T. Bartz-Beielstein, M. Preuss, and C. Stoean. *A Support Vector Machine-Inspired Evolutionary Approach for Parameter Setting in Metaheuristics.* Cologne University of Applied Science. 2009.

[37] Th. Stützle, M. Lopez-Ibanez, P. Pellegrini, M. Maur, M.M. de Oca, M. Birattari and M. Dorigo. *Parameter Adaptation in Ant Colony Optimization.* IRIDIA - Technical Report Series, Belgium. 2010

[38] P. Toth and D. Vigo. *The Vehicle Routing Problem*. SIAM. 2002

[39] E.G. Talbi. *Metaheuristics: From design to implementation*. John Wiley & Sons. 2009

[40] B. van Veen, M. Emmerich and Th. Back. *Solving the Dynamic Vehicle Routing Problemwith Time Windows using Ant Colony Optimization.* Leiden University. 2013

[41] J.M. van Ast, R. Babuska and B. De Schutter. *Ant colony learning algorithm for optimal control.* In: Interactive Collaborative Information Systems (R. Babuska and F.C.A. Groen(Eds.), vol. 281 of Studies in Computational Intelligence, Berlin, Germany: Springer, p55–182. 2010.

[42] I. Voutchkov and A. Keane. *Multi-objective optimization using surrogates.* In: Adaptive Computing in Design and Manufacture ACDM. 2006.

[43] S. Wink, Th. Back and M. Emmerich. *A Meta-Genetic Algorithm for Solving the Capacitated Vehicle Routing Problem*. Leiden University. 2011

[44] Z. Yuan, M.M. de Oca, M. Birattari, and Th. Stützle. *Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms.* IRIDIA - Technical Report Series, Belgium. 2011

[45] M. Zaefferer, Th. Bartz-Beielstein, M. Friese, B. Naujoks and Ol. Flasch. *Multi-Criteria Optimization for Hard Problems under Limited Budgets*. In: GECCO Companion ’12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion, Philadelphia, Pennsylvania, USA, p1451-1452. 2012.

[46] M. Zaeferer, Th. Bartz-Beielstein, B. Naujoks, T. Wagner, and M. Emmerich. *A Case Study on Multi-Criteria Optimization of an Event Detection Software under Limited Budgets.* In: Evolutionary Multi-Criterion Optimization 7th International Conference, EMO. 2013.

Websites:

[47] CPU Benchmarks, accessed: 26.01.2012, website: http://www.cpubenchmark.net

[48] SWIG, accessed: 26.10.2012, website: http://www.swig.org/

[49] R.NET, accessed: 02.11.2012, website: http://rdotnet.codeplex.com/

[50] R(D)COM, accessed: 25.10.2012, website: http://cran.r-project.org/other-software.html

[51] Xtreme Route. accessed: 12.01.2013, website: http://www.xtremeroute.com/

Appendices

Appendix 1

Extensive comparison tuning approaches [17]

		Performance		Seeds		Parameters		Problems	
Described in Section	Method	Quality	Speed	Success	Stability	Tolerance	Tuneability	Applicability	Fallibility
7.1	Latin-Square [29]	- -	- -	- -	□	- -	- -	- -	- -
	Taguchi Orthogonal Arrays [30]	- -	- -	- -	□	- -	- -	- -	- -
7.1.1	CALIBRA [40]		□		□	+	+	- -	- -
	Empirical Modelling of Genetic Algorithms [29]		□		-	+	+	- -	- -
7.2	Sequential Experiment Designs [61]	-	-	-	-	□	□	□	□
	François-Lavergne [43]	-	-	-	-	□	□	□	□
	Logistic Regression [42]	-	-	-	-	□	□	□	□
	ANOVA [34]	-	-	-	-	□	□	□	□
	Design of Experiments with Regression Tree [62]	-	-	-	-	□	□	□	□
7.2.1	Coy's Procedure [44]		+		-	□	□	□	□
	Sequential Parameter Optimization (SPO) [36]		++		-	++	++	+	+
	SPO + OCBA [45]		++		-	++	++	+	+
7.3	Interactive Analysis [47]		-		- -	□	+	- -	- -
	Ranking and Selection [48]		-		- -	□	+	- -	- -
	Multiple Comparison Procedures [49]		-		- -	□	+	- -	- -
	Sequential indifference-zone selection [50]		-		- -	□	+	- -	- -
	Racing [35]		-		- -	□	+	- -	- -
	F-RACE [16]		-		- -	□	+	- -	- -
7.3.1	Iterative F-RACE [39]		+		- -	++	+	- -	- -
7.4	Meta-Plan [52]		++		-	- -	- -	- -	- -
	Meta-Algorithm [32]		++		-	- -	- -	- -	- -
	Meta-GA [63]		++		-	- -	- -	- -	- -
	Meta-GA + Racing [5]		++		-	- -	- -	- -	- -
	FocusedILS [53]		++		-	- -	- -	- -	- -
	Meta-ES [64]		++		-	- -	- -	- -	- -
	Meta-CMA-ES [25]		++		-	- -	- -	- -	- -
	OPSO [65]		++		-	- -	- -	- -	- -
7.4.1	Local Unimodal Sampling [57]		+		-	- -	- -	□	□
	REVAC [66]		+		-	++	+	- -	- -
	REVAC ++ [25]		++		- -	++	+	□	□
7.4.2	M-FETA [58]	+	+	+	+	□	□	++	++
	Performance Fronts [59]	+	+	+	+	-	-	+	+

Algorithms that span multiple columns can optimize/give information on one of the columns at the time

++ excellent quality + good quality □ reasonable quality - poor quality - - very poor quality

Appendix 2

SPOT interface to HGA

```
rm(list=ls());
require(SPOT)
setwd("D:/vrp/Vehicle Routing Problem/ConsoleApplication/")

ptm <- proc.time()
callString1 <- paste("csc ConsoleProgram.cs /r:Logic.dll /r:Data.dll")
callcs <-system(callString1, intern= TRUE)

#now define a target function for SPOT
tGA <- function(pars){
        #the tuned parameters:
         mu<- round (pars [1])
    lambda<-round (pars[2])
    ts<-round (pars [3])
    pm<-round (pars [4])
    ro<- round (pars [5])
    mo<- round (pars [6])
    io<- round (pars [7])
    ss<- round (pars [8])


  #now start ES and record used time as well as best function value
        ti<-as.numeric(system.time({

            callString2 <- paste("ConsoleProgram", mu, lambda, ts, pm, ro, mo, io, ss )
            y <-system(callString2, intern= TRUE)
                    print(c(mu, lambda, ts, pm, ro, mo, io, ss ))}
             ))[1]

        return(c(as.numeric(y),ti)) #Y1 is the best value reached, Y2 is the time used}

#define region in which parameters of GA are tuned
roi<spotROI(c(3,2,2,1,0,0,0,0),c(10,5,17,4,1,1,1,1),
type=c("INT","INT","INT","INT","INT","INT","INT","INT"))

#define further settings for SPOT
config <- list( alg.func=tGA, #target of SPOT
            alg.roi=roi, #region of interest of SPOT
             seq.predictionModel.func="spotPredictForrester", #a kriging surrogate model
            #seq.predictionModel.func="spotPredictRandomForest",
            #seq.predictionModel.func="spotPredictEarth",
            seq.predictionOpt.func="spotParetoOptMulti", #optimize surrogate models
            #seq.predictionOpt.func="spotPredictOptMulti",
            #seq.predictionOpt.method="sms-emoa",#optimize surrogate model with sms-emoa
            seq.predictionOpt.method="nsga2", #optimize surrogate model with nsga2
            seq.predictionOpt.budget=2000, #1000 evaluations of surrogate models
            seq.predictionOpt.psize=20, #population size of sms-emoa or nsga2
            io.verbosity=3, #create some text output
            spot.ocba=T, #no optimal computational budget allocation
            spot.fileMode = T, #  RES, DES, BST files are created
            auto.loop.nevals = 100, # the number of times the GA is run
```

```
            seq.design.oldBest.size=2, #number of old points to repeat in each step
            seq.design.new.size=1, #number of new points to evaluate
            seq.design.size = 2000, #large candidate design evaluated on surrogate
            init.design.size = 20,  #number samples in initial design
            seq.design.maxRepeats = 3, #maximum number of evaluations of a design point
            init.design.repeats  = 2, #number of evaluations of each initial design point
            spot.seed = 125)
            #init.design.func = "spotCreateDesignLhd",



#run SPOT       (this might take some time, depends on your machine)
res <- spot(spotConfig=config)

#look at results (raw)
res$alg.currentResult

#pareto optimal parameters found (pareto set):
res$mco.par

#pareto front:
res$mco.val

proc.time() - ptm


rm(list=ls());
require(SPOT)
#setwd("D:/vrp/Vehicle Routing Problem/ConsoleApplication/")

ptm <- proc.time()
```

Appendix 3

SPOT interface to MACS-DVRPTW

```
#now define a target function for SPOT
tGA <- function(pars){
          #the tuned parameters:
     iNumAnts <- round (pars [1], digits = 1)
     iRho<- round (pars[2], digits = 1)
     iAlpha<-round (pars [3], digits = 1)
     iBeta<-round (pars [4], digits = 1)
     iQZero<- round (pars [5], digits = 1)
     iPheromonePreservation<- round(pars [6], digits = 1)
     #io<- round (pars [7])
     #ss<- round (pars [8])
    #signif(x, digits = 6)

   #now start ES and record used time as well as best function value
          #ti<-as.numeric(system.time({


setwd("/home/chill/Ma/Bridge/") # to be able to write the desired values in the txt file
```

```
callString1 <- paste("gcc -o write Bridge.c")
call1 <-system(callString1, intern= TRUE)

callString2 <- paste("./write", iNumAnts, iRho, iAlpha, iBeta, iQZero, iPheromonePreservation)
call2 <-system(callString2, intern= TRUE)

setwd("/home/chill/Ma/") # reason for a different folder is incompatibility with the above Bridge.c
callString3 <- paste("make")
callcs <-system(callString3, intern= TRUE)

callString4 <- paste("./main")
y <-system(callString4, intern= TRUE)


                    print(c(iNumAnts, iRho, iAlpha, iBeta, iQZero, iPheromonePreservation))
            #))[1]

        #return(c(as.numeric(y),ti)) #Y1 is the best value reached, Y2 is the time used
    #return(as.numeric(y)) }
    return(y) }

#define region in which parameters of Macs-DVRTW are tuned
roi <- spotROI(c(5,0,1,1,0,0),c(100,1,5,5,1,1),type=c("INT","FLOAT","INT","INT", "FLOAT",
"FLOAT"))
#roi <- spotROI(c(10,1),c(100,5),type=c("INT","INT"))

#define further settings for SPOT
config <- list( alg.func=tGA, #target of SPOT
          alg.roi=roi, #region of interest of SPOT
                #seq.predictionModel.func="spotPredictForrester", #a kriging surrogate model
          seq.predictionModel.func="spotPredictRandomForest",
          #seq.predictionModel.func="spotPredictEarth",
          #seq.predictionOpt.func="spotParetoOptMulti", #optimize surrogate models
          seq.predictionOpt.func="spotPredictOptMulti",
          seq.predictionOpt.method="cmaes",
          #seq.predictionOpt.method="sms-emoa",#optimize surrogate model with sms-emoa
          #seq.predictionOpt.method="nsga2", #optimize surrogate model with nsga2
          seq.predictionOpt.budget=2000, #2000 evaluations of surrogate models
          seq.predictionOpt.psize=20, #population size of cmaes
          io.verbosity=3, #create some text output
          spot.ocba=F, #no optimal computational budget allocation
          spot.fileMode = T, #  RES, DES, BST files are created
          auto.loop.nevals = 200, # the number of times the algorithm is run
          seq.design.oldBest.size=2, #number of old points to repeat in each step
          seq.design.new.size=1, #number of new points to evaluate
          seq.design.size = 2000, #large candidate design evaluated on surrogate
          seq.design.maxRepeats = 4, #maximum number of evaluations of a design point
          init.design.func = "spotCreateDesignLhd",
          init.design.size = 30,  #number samples in initial design
          init.design.repeats  = 3) #number of evaluations of each initial design point
          #spot.seed = 125)

#run SPOT       (this might take some time, depends on your machine)
res <- spot(spotConfig=config)
```

```
#look at results (raw)
res$alg.currentResult

#pareto optimal parameters found (pareto set):
res$mco.par

#pareto front:
res$mco.val

proc.time() - ptm
```