INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Optimization of the Multi-Depot Vehicle Routing Problem: an Application to Logistics and Transport of Biomass for Electricity Production

Tiago Caperta Maia Caldeira

Dissertação para obtenção do Grau de Mestre em
Engenharia Mecânica

Presidente:	Prof. João Rogério Caldas Pinto
Orientador:	Prof. João Miguel da Costa Sousa
Co-Orientador:	Prof. Carlos Augusto Santos Silva
Vogal:	Profª. Alexandra Bento Moutinho

Outubro de 2009

This work reflects the ideas of its authors, which might eventually not coincide with IST's.

Acknowledgements

First and foremost, I would like to express my gratitude towards Professors João Sousa and Carlos Silva for the constant support, advice and guidance offered throughout the development of this work but also for giving me the opportunity to broaden my knowledge on this subject.

I would also like thank the authors of the references quoted, for all the work done in their respective fields.

Two final acknowledgements should go to Quan Quach, Daniel Sutoyo and everyone else at http://blinkdagger.com/ for the excellent MATLAB tutorials and also to Steve Hoelzer for the progress bar used in the application developed.

Abstract

The work was developed with the aim of creating an application that would help solve both the energy dependency and waste collection problems in the Azores islands. It contributes to the specific issue of optimizing the transport of biomass generated through human activities to energy production plants that use it as fuel, by attempting to reduce the amount of time and effort consumed in said transport.

Even though the application was developed with this aim in mind, it is able to solve any problem of the Multi-Depot Vehicle Routing Problem (MDVRP) class by using three different and independent algorithms based on metaheuristics, two of them adapted from existing literature and a new one developed specifically for this work.

Results using all three algorithms on both benchmark instances (p01, p04, p07, p09 and p13 of (Díaz 2007)) and a specific problem related to the Azores islands are presented and compared.

Keywords: Azores, Biomass, Multi-Depot Vehicle Routing Problem, Metaheuristics

Resumo

O trabalho apresentado foi desenvolvido com o objectivo de criar uma aplicação que simultaneamente respondesse a dois problemas relativos ao arquipélago dos Açores, a dependência energética e a recolha de resíduos. Contribui para ambos ao tentar optimizar o transporte da biomassa gerada por actividades humanas para centrais energéticas que usem este tipo de combustível, tentando reduzir o tempo e custos associados ao mesmo.

Mesmo tendo sido este o principal propósito que levou à criação da aplicação, esta é capaz de resolver qualquer problema da classes *Multi-Depot Vehicle Routing Problem* (MDVRP) utilizando três algoritmos distintos e independentes baseados em meta heurísticas, dois dos quais foram adaptados de literatura existente, tendo o terceiro sido desenvolvido especificamente para este trabalho.

Os resultados da utilização dos três algoritmos tanto a problemas de referência (p01, p03, p07, p09 e p13 de (Díaz 2007)) como a um problema local relativo aos Açores são apresentados e comparados.

Palavras-chave: Açores, Biomassa, *Multi-Depot Vehicle Routing Problem*, Meta heurísticas

Table of Contents

List of Figures

List of Tables

1. Introduction

As an introduction to the following work, a brief contextualization of the current energy scenario will be made, along with the specificities of the area for which it was developed, its goals and contributions.

1.1. Energy: A Historic Perspective

The first recorded use of the word energy was made by Aristotle in his work *Nicomachean Ethics* in the 4th century BC, but the concept had been around since the dawn of the human era. The contested formula (Basalla 1980) “energy = progress = civilization” is still very much true today as it was then.

The earliest uses of energy by humans were solely based on the conversion of nutrients from food into raw power used by each individual’s muscles, but soon our intellect lead us to more rational applications, the first of which was utilizing the combustion of wood and other materials to provide energy for heating, cooking, lighting or early furnaces (Barbour, et al. 1982).

Before the Industrial Revolution, mankind still relied heavily on their own muscular strength and those of animals, as well as making use of wind and water power in mills and later in manufacturing plants. Water mills were especially used, as Europe had abundant sources of this energy, causing riverside towns and settlements to appear and flourish through the use of this resource (Williams 2006). With the development of steam power, the dependency on a close water source was eliminated and new factories and plants sprung up everywhere, a phenomenon that was exacerbated by the progresses in electricity production and transportation, leading to cheap and accessible energy for everyone.

The wonders of the new electrical age including the development of new means of transport were soon coveted by all, leading to a rise in the cost of fossil fuels (coal and oil). This was somewhat offset by the continual improvements made to engines and power generating equipment but ultimately not even they could cope with the generalization of electric power and vehicles.

With the discovery by Marie Curie of radioactivity followed by the use of nuclear fission in World War II, a new source of energy was soon applied to electricity production, nuclear energy. The enormous amount of energy generated and the abundance of fuel, compared to oil and coal, lead to an initial spread of nuclear plants in the developed world. More recently though, environmental and safety concerns have all but halted the construction of new nuclear plants (Climate Group 2006).

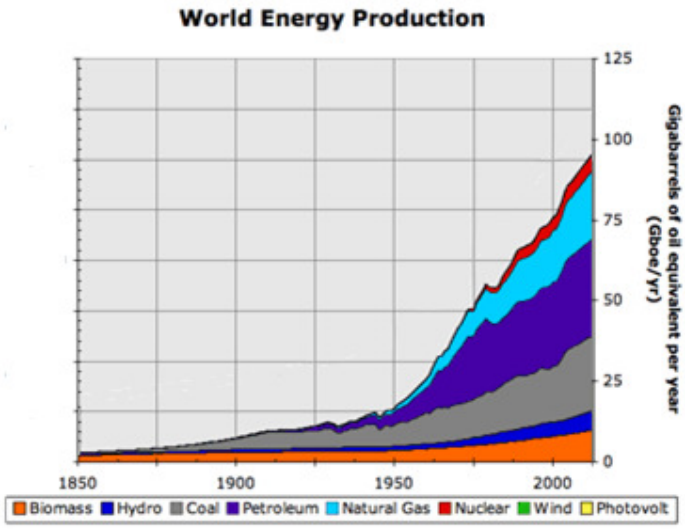

Figure 1-1 Total energy usage through history – based on (Cobb 2007)

1.2. Renewable Energies

As can be inferred from the previous section, the foundations of the current global economy lie in the concept of accessible, cheap energy, available through the consumption of fossil fuels, but the large growth that has been observed in demand in the last few decades (see Figure 1-1 Total energy usage through history – based on (Cobb 2007)Figure 1-1) and the increasing scarcity of these fuels will inevitably lead to a rise in prices. Therefore, it is only natural that we should look to other, economically viable, sources of energy that will support the current economic paradigm. In this context, renewable energies appear as clean, limitless alternatives to fossil fuels and nuclear energy proliferation.

The two main sources of renewable energy in use today are hydroelectric power, which has been in use since before the Industrial Revolution, and biomass, even though its major contribution is still heating, through burning of wood (Kanellos 2008). Wind and solar power are growing, but still only represent a very small fraction of the global energy supply (Figure 1-2 shows recent energy production distribution).

Hydroelectric power, despite being a major player at the moment, does not have a very large growth potential, as many of the developed countries have already tapped the available sources or are unwilling to do so due to environmental and social concerns (World Nuclear Association 2009). On the other hand, biomass has always had a large growth potential as our society produces a great deal of waste that can be effectively turned into energy with the correct technology. The challenge lies in utilizing this waste to produce electricity or other fuels (e.g. biofuels) in a cost effective manner. In the biomass case, it has been shown that the majority of costs are incurred during transportation due to the fact that, with current technology, larger (and bulkier) amounts of mass are required to produce energy when compared to traditional fossil fuels (Rentizelas, Tatsiopoulos and Tolis 2008).

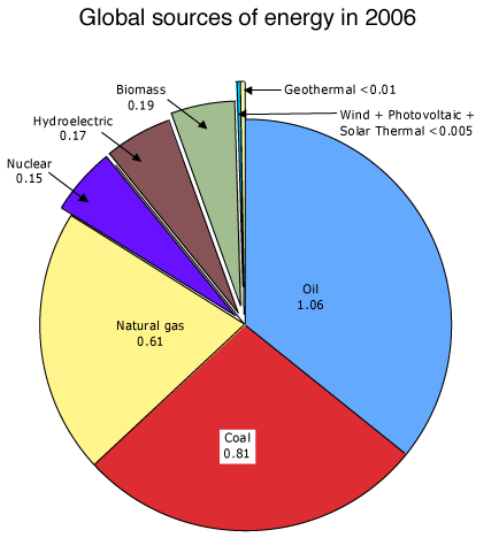

Figure 1-2 Global energy contributions in CMO[1] – from (Kanellos 2008)

1.3. The Azores Case

The case study for this work is the Azores archipelago, which is located in the Atlantic Ocean and is part of the Portuguese Republic. It has a total surface area of 2346 km^2 and is composed of 9 inhabited islands (and a few more uninhabited ones) located around 33º N 28º W as shown in Figure 1-3.

Figure 1-3 Map of Azores – from (Maps of the World n.d.)

The archipelago is simultaneously facing two issues that might be dampened by the use of renewable energy sources, specifically biomass. On one hand, the dependency on imported fossil

[1] Cubic Mile of Oil

fuels for the archipelago's energy needs, and on the other hand, the problem posed by the waste generated on the islands that takes up precious land space or requires exportation (Brito, et al. 2007). Also, increasing the use of biomass on the islands will aid Portugal in complying with the quotas established by the UE (in 2010, 39% of electricity produced should be renewable) (INETI 2005).

1.4. Goals

The main goal of the following work is to use a variety of metaheuristics to solve the Multi-Depot Vehicle Routing Problem (explained in Section 2) and to incorporate them in a MATLAB application with the purpose of establishing optimal or near optimal routes for the vehicles that will transport the biomass from various collection sites across the islands under study (essentially S. Miguel, the biggest island) to the various power stations that may exist.

The application developed should have a user friendly environment and easy to use data importation and exportation functions so it can be operated by users inexperienced with the programming behind it.

1.5. Contributions

The main contribution of this work is a fully functional application for solving the Multi-Depot Vehicle Routing Problem using three different metaheuristics, whether it is applied to biomass or any other problem.

Scientific contributions were also made to the Tabu Search and Genetic algorithms used such as the use of different initial solutions and a new sub-cycle parameter, as well as the development of a new Ant Colony Optimization algorithm applied to the above mentioned routing problem, which has never been done to the best of our knowledge.

Following this work, an article will be submitted to the WCCI conference of 2010 relating to evolutionary computation.

2. The Multi-Depot Vehicle Routing Problem

In order to obtain the most cost/time efficient manner of transporting biomass to more than one power plant, we will need to solve a Multi-Depot Vehicle Routing Problem (MDVRP). The problem will be defined in the next subsection and two heuristics that have been used to solve it in the literature will also be presented.

2.1. Definition

The problem that we propose to solve is the following: Having a fleet of vehicles based on one or more biomass plants, what is the most economically advantageous way of picking up the waste from various collection sites and return to the plant? (Xu and Kelly 1996)

In the literature, this problem is referred to as the Multi-Depot Vehicle Routing Problem (MDVRP). It is a non-linear programming problem that can be seen as a generalization of the Traveling Salesman Problem (TSP) differing in the fact that we now have various salesmen and each site must be visited by one and only one of them, while minimizing the total traveling cost of all. In Figure 2-1 a possible VRP solution with only one depot is shown.

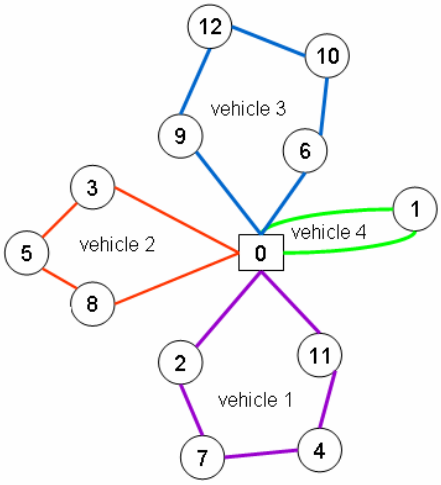

Figure 2-1 Possible final solution of a VRP – from (Medaglia and Gutiérrez 2005)

The formal definition of the VRP (with only one depot or plant) is the following. Let G = (V, A) be a graph where V = {0, 1, ..., N} is a set of nodes and A is a set of arcs. Node 0 is the plant and the remaining nodes are the collection sites[2]. From the plant, a set of K vehicles with capacity Q^K leave with the purpose of collecting a certain quantity q_i from each collection site, knowing that a trip between any vertex (regardless of being a plant or a collection site) incurs a non-negative cost c_{ij}, which will usually represent a distance/time that must be covered/spent (therefore this matrix is usually symmetric - $c_{ij} = c_{ji}$) being C^k the maximum total cost that a vehicle K can incur (Renaud, Laporte and Boctor 1995).

[2] Henceforth the location from where vehicles depart will be always referred to as the plant and the vertexes they visit will be referred to as collection sites.

With that said, our main goal is then to:

Minimize $$\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=1}^{K} c_{ij} \cdot X_{ij}^{k} \quad (1)$$

Subject to $$\sum_{i=0}^{N}\sum_{j=0}^{N} X_{ij}^{k}\, q_i \leq Q^k \qquad 1 \leq k \leq K \quad (2)$$

$$\sum_{i=0}^{N}\sum_{j=0}^{N} X_{ij}^{k}\, c_{ij} \leq C^k \qquad 1 \leq k \leq K \quad (3)$$

$$\sum_{j=1}^{N} X_{0j}^{k} = \sum_{j=1}^{N} X_{j0}^{k} \leq 1 \qquad 1 \leq k \leq K \text{ and } i = 0 \quad (4)$$

$$\sum_{k=1}^{K}\sum_{j=1}^{N} X_{ij}^{k} \leq K \qquad i = 0 \quad (5)$$

where X_{ij}^{k} is a binary variable that represents whether or not the link between sites *i* and *j* will be used by vehicle K.

In this case, the cost function (1) only represents the total distance or time covered or spent by all the vehicles, but it could easily be changed to contemplate other factors, the most obvious being total fleet size.

Constraint (2) illustrates that a vehicle can only transport as much as its maximum capacity, (3) is similar but relative to the maximum distance or time that can be covered/spent by a single vehicle, (4) forces all routes to start and finish at the plant and (5) limits the number of routes to the total number of vehicles.

In addition to the presented constraints many others could be included, such as specific time windows during which a vehicle can or cannot visit a certain collection site, or the insertion of more plants from which they can leave and arrive at (turning it into a MDVRP). These and other changes only contribute to make a hard problem even harder, granting the VRP the status of NP-hard problem (non-deterministic polynomial time hard) (Garey and Johnson 1990).

For a problem of this degree of complexity it is usually not possible to obtain optimal solutions except in very small cases (see (Laporte 1992)). For this reason, a number of heuristics that obtain satisfactory results in feasible time frames, hopefully near the global optimum in performance terms, have been developed.

2.2. Classic Heuristics

The two classic heuristics that will be presented have in common the attributes of being fast, reliable methods for obtaining good solutions for the single depot VRP. To apply them to the multi-depot version, all the collection sites are clustered around their closest plant and the heuristic is applied for each plant as if it was a single depot VRP[3].

While the solution is satisfactory it will most probably not be the global optimal or close to it, so these heuristics are used extensively to provide an initialization for more thorough approaches.

2.2.1. Clark Wright Savings Algorithm

The Clarke Wright Savings Algorithm (Clark and Wright 1964) can be applied to Vehicle Routing Problems with only one plant and an arbitrary number of vehicles.

[3] Therefore, in the following subsections the heuristics will be described as if we were solving a single depot VRP.

The algorithm begins by giving each collection site his own route, in other words, N simple routes are created consisting of the trip from plant to collection site and back again. Following this, a savings list is compiled, consisting of the amount that the total cost would be reduced if two existing routes would be merged. As an example, by merging the route {Plant -> Site 1 -> Plant} with the route {Plant -> Site 2 -> Plant} as seen in Figure 2-2 we would obtain {Plant -> Site 1 -> Site 2 -> Plant}, eliminating the cost of the trips {Site 1 -> Plant} and {Plant -> Site 2} and adding a new trip (with matching cost), {Site 1 -> Site 2}. The total saving that would be incurred from this merge is then placed in the list which, when completed, is sorted in decreasing order. Attempts are then made to implement the merging starting from the top of the list.

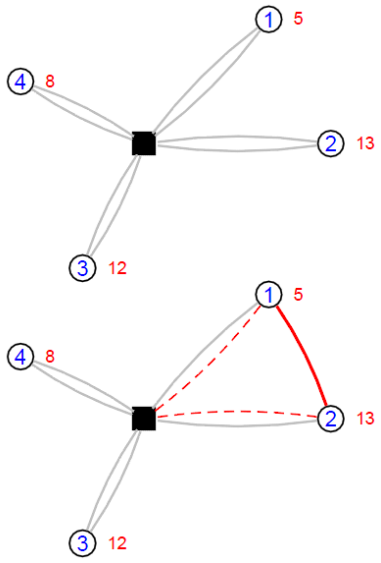

Figure 2-2 Possible heuristic evolution from the initial state – from (Battarra, Baldacci and Vigo 2007)

In an algorithmic form:

Clark Wright Savings Algorithm

1. *Create individual routes, starting from the plant and visiting a single collection site*
2. *Build savings list for every combination of sites and following* $s_{ij} = c_{i0} + c_{0j} - c_{ij}$.
3. *Set the list in decreasing order*
4. *Starting at the top of the list, attempt to implement merge*
    - *I. If saving is positive proceed, else go to 5.*
    - *II. Make sure sites in question are at the edges of routes (they must still have a direct link to a plant), if not go to V.*
    - *III. Make sure the merge does not violate constraints (capacity, length, etc), if it does go to V.*
    - *IV. Implement merge*
    - *V. Move to next list member and return to I. If the end of the list has been reached go to 5.*
5. *Terminate algorithm*

There is another version of this algorithm referred to in the literature (sequential version) but it has been proven that this, parallel version, obtains better results (Toth and Vigo 2002). The main advantages of this heuristic are its speed and easy implementation, but the results are normally within 5% of best know solutions for benchmark problems. In a worst case scenario, the algorithm will return a solution that will be $\log(2n) + 1$ times larger than the global optimum.

2.2.2. Improved Petal Heuristic

The improved petal heuristic was developed in (Renaud, Boctor and Laporte 1996-B) as an attempt to respond to situations where the user desires a good solution for a VRP problem but does not have the luxury of time for extensive computations using, for example, metaheuristics or branch & cut algorithms. In short, it consists of generating a large group of feasible routes by using the sub-heuristics one-petal and two-petal and then using a set partitioning algorithm to choose some of them.

The algorithm starts off by obtaining the polar coordinates of all the N collection sites, in relation to the plant which is centered at the origin. The sites are then sorted in increasing order of polar angle, with ties being broken by placing the closest site before the furthest (distance measured relative to the plant). We can then start the petal construction cycle, starting with $i = 1$:

1. Set $i = i + 1$. If $i > N$ terminate algorithm.
2. Build a back and forth route between the plant and site i. Record it and its cost.
3. Set $j = i + 1$ and build a route between the plant and sites i and j. If it is infeasible go to Step 5, else record the route and its cost.
4. Set $j = j + 1$ and build a route between the plant and all sites $\{i, \ldots, j\}$ using the one petal heuristic if the sum of mass to collect is smaller than the capacity of one vehicle. If it is not or if there is no feasible route go to Step 5. If a feasible route is obtained, record it and its cost and repeat this step.
5. Build a route between the plant and all sites $\{i, \ldots, j\}$ using the two-petal heuristic if the sum of mass to collect is smaller than the capacity of two vehicles. If it is not or if there is no feasible petal[4] obtainable go to Step 6. If a feasible petal is obtained, record it and its cost and repeat this step.
6. Apply a dominance test by starting at the last petal created, removing the last site inserted in it (with accompanying change in cost) and comparing its new cost with the previous petal. If it is smaller, overwrite the previous petal with this one as it is dominated. Repeat for remaining petals constructed in steps 4 and 5. Return to step 2.

Next, we must look at the way in which the one-petal and two-petal heuristics operate in order to produce routes.

One-petal

The extended description of the one-petal heuristic can be found in (Renaud, Boctor and Laporte 1996-A) but in short it consists of 3 phases:

- Initialization – The route is initialized by first selecting the northernmost site[5] and then adding the ones to the east of it. Following this, the easternmost site is selected and then the southernmost ones are added. The same happens for South/West and then West/North. This heuristic constructs an envelope of vertexes as can be seen in Figure 2-3.

[4] A petal is defined as a group of routes, in this case, 2.
[5] Even though the word site is used, in this and the following phases of the heuristic no distinction is made between collection sites and plants

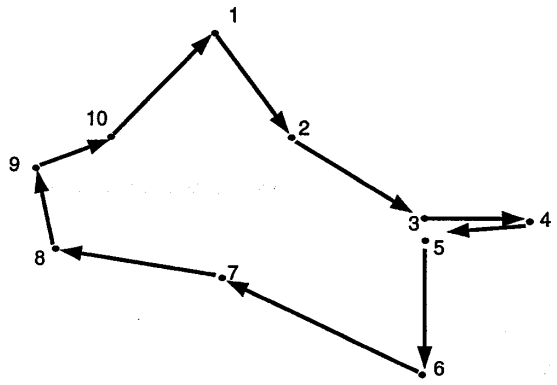

Figure 2-3 Route initialization example – from (Renaud, Boctor and Laporte 1996-A)

The heuristic can be turned around (going anti-clockwise instead of clockwise) but the solution, while being different, might or might not be better.

- Insertion of remaining sites – Due to the nature of the previous phase, interior sites might be left out and must be inserted in the route.

  In order to do so, a cheapest insertion criteria is used, namely we compute the insertion costs of every remaining site in each possible location, saving the smallest for each site. Then, we select the site with the cheapest insertion ratio (costs of traveling to and from the new site divided by cost link that will be broken) and place it in the location found previously.
- Improvement – In the final phase, the plant is found within the route and it is re-ordered. Additionally, a local optimization algorithm which will be described later in Section 4.2.1.1 is applied to the tour, after which it is saved along with the cost.

Two-petal

The two-petal heuristic is described in detail in (Renaud, Boctor and Laporte 1996-B) but will also be explained here in short.

The aim of this heuristic is to construct two different routes that might intercept each other, as this is often found in optimal solutions. The initialization is done by choosing the two sites that are furthest apart from each other and assigning them back and forth routes to the plant. Following this, we repeatedly apply a cheapest insertion criterion to place the remaining sites in either of the routes.

However, to prevent the early formation of unbalanced routes that might condition the insertion of later sites, an α coefficient is defined ($\alpha = 2Q - \sum q_i$), where Q is the capacity of one vehicle and q_i is the quantity supplied by site *i*. This coefficient will be small if the current problem is tightly constrained and larger if not. We then enforce that the difference in transported load from one route to the other must not exceed α, therefore creating balanced routes only when they are needed.

Despite this, the α criterion can be ignored if all insertions failed. If, even after ignoring it, certain sites remain un-inserted, a series of exchanges between both routes (described in detail in Section 4.2.1.1) are attempted and, if any is successful, the insertions are re-tested. Furthermore, whenever the number of sites in any of the routes reaches a multiple of 5, local optimization is applied to it.

When all sites have been inserted in either of the routes, each of them is re-optimized using the one-petal heuristic described above.

When the main body of the algorithm terminates, which means that all possible petals have been formed, a set partitioning problem needs to be solved in order to choose exactly which routes will be used in the solution. For this part of the algorithm, binary programming is used to minimize the total cost of the selected petals while ensuring that all sites are present in the solution exactly once.

2.2.3. Heuristic Comparison

The Improved Petal Heuristic provides solutions of generally higher quality than its Clark Wright counterpart. However it does have its drawbacks. It is computationally more intense, it is not deterministic (due to the local optimization and exchanges in two-petal formation) and requires the data to be supplied in the form of coordinates, in order to apply the one-petal heuristic and obtain the polar coordinates. This last constraint makes it infeasible to apply if the data is supplied as a time or distance matrix for example.

3. Metaheuristics

Besides specific heuristics developed for the VRP, attempts have been made to apply general metaheuristics to this problem. In the following subsections three popular metaheuristics will be described in general but the way in which they are applied to the problem will be left for Section 4.

3.1. Tabu Search

The word “tabu” comes from a Polynesian dialect where it means “something that cannot be touched because it is sacred”, but the meaning given today is more in line with “a prohibition imposed by social custom as a protective measure” (Merriam-Webster 2009).

The roots of the Tabu Search algorithm are presented in the seventies but the version that is commonly used today was presented by Fred Glover in 1986, having since then been subjected to innumerous proposals of alterations and adaptations.

Before Tabu Search, the search for optimality for a given problem was done mostly with classic hill climbing techniques, meaning that the algorithms would, from their current position, attempt to find the best possible improvement. The problem with this type of approach is the high probability of reaching a local maximum (or minimum) from which the algorithm cannot escape, returning a solution that can be far from optimal as can be seen in Figure 3-1.

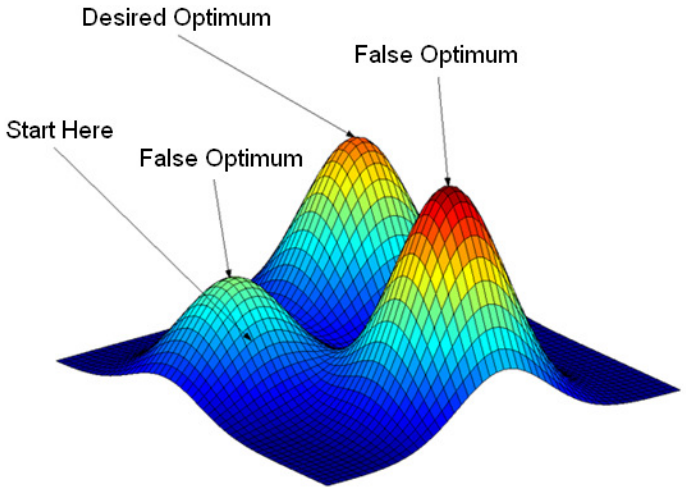

Figure 3-1 Hill climbing algorithms will probably become stuck in one of the false optimums, while Tabu Search might reach the desired one – from (Bridger 2007)

The answer to this problem given by the Tabu Search algorithm comes in the form of allowing, if no improvement can be found in the neighborhood, the solution to evolve to worse performances. In this way, it can move from the local optimum to explore new areas of the solution space. The issue

with this approach is the creation of cyclic moves, where the same points are visited in succession (Stidsen 2008).

To solve this, the tabu list that gives the metaheuristic its name is implemented. It contains a memory of the previously visited solutions (or only the recently visited ones) and does not allow the algorithm to evolve to them again. Therefore, as per the definition of the tabu word, the list is protecting the algorithm from cyclic moves and/or getting trapped in a certain region.

The tabu list can also be seen as the algorithm's memory. This sets the algorithm apart from memory less methods, such as the Genetic Algorithms presented below or Simulated Annealing, but it is also not exactly the same as hard-coded memory of Branch & Bound or Branch & Cut approaches because, for one, the tabu list has a limited size (which can be constant or not) and so it acts like an actual biological memory, where older events are replaced by new ones. On the other hand, a Tabu Search implementation usually comes with an aspiration criterion. This makes sure that, under special circumstances, a tabu move may be accepted. The most obvious circumstance is when said move generates a solution better than any found until then, but other improvement strategies may be used.

The general sequence of operations in a Tabu Search metaheuristic will be the following:

General Tabu Search Algorithm

1. *Start the problem with a feasible solution and an empty tabu list.*
2. *Search the neighborhood for other possible solutions.*
3. *Eliminate from these the ones that are in the tabu list, keeping aspiration criterion in mind.*
4. *Choose the best solution between the ones found.*
5. *Include the move in the tabu list.*
6. *Go to 2.*

The termination criteria can vary, but may include a fixed number of iterations, iterations without improvement, computing time, among others.

Since its first implementation, Tabu Search has been applied to a large array of complex problems. From logistics to biomedical analysis and from financial analysis to space planning, the scope is nearly endless (Glover and Laguna 1997).

3.2. Genetic Algorithms

The concept of applying the principles of Darwin's evolution theory to the resolution of problems had already been present since the fifties and sixties, especially after "Evolution Strategies" (Rechenberg 1973), but it was in 1975, with the publishing of *Adaptation in Natural and Artificial Systems* (Holland 1975), that it gained a large amount of recognition. A Genetic Algorithm consists then in the application of the principles of evolution and adaptation to the environment that are present in every species to optimization problems, in order to reach the desired goal through survival of the fittest.

The biological chromosome consists of a series of genes which are necessary to code specific proteins. Simplistically we can view genes as a coded version of an individual's traits, intelligence, skin and eye color, height or even sex. During conception, the parents' chromosomes mix and the offspring is generated with some characteristics from each of them. Mutation can also occur, which is the result

of coding errors and can result in the offspring possessing a trait that was not present in either of its parents.

The main difference between Genetic Algorithms and other optimization techniques (such as the Tabu Search described in Section 3.1) is the use of a population of solutions instead of a single one. This allows for more exploration during a single iteration, as many possibilities are being evaluated at once, and there is also more exchange of information between the solutions evaluated in parallel.

The algorithm is started by obtaining an initial population of chromosomes (the name given to solutions) randomly or incorporating some previous knowledge of the problem. These are then evaluated according to a fitness function that measures their adaptability to the problem's environment, which corresponds to the cost function of the problem being optimized. From there, some are selected to become parents (usually the fitter individuals have a higher chance of becoming parents) of the next generation and mate. This reproduction, or crossover, is done by creating two or more new chromosomes that contain part of the parents' genetic material. Doing this for all pairs of selected parents will result in a new generation and the process can then be repeated, with some descendants possibly being identical to the best parents to preserve solution quality (elitism). Mutation, just like in natural organisms, can occur and operate a minor change on a chromosome. This usually occurs randomly on a low percentage of chromosomes and is useful to explore new regions of the solution space that might not have been covered by the population's current genetic material (Marczyk 2004). A simple diagram representing the evolution of a Genetic Algorithm is shown in Figure 3-2.

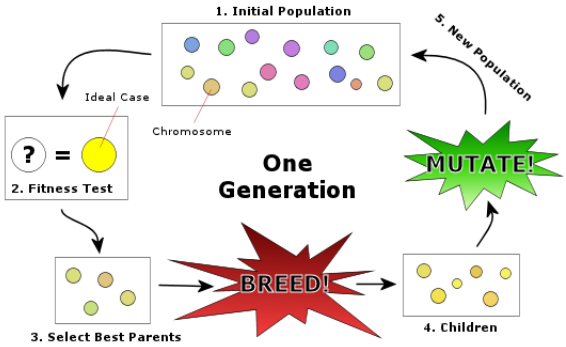

Figure 3-2 Visual representation of a Genetic Algorithm – from (Pote 2006)

One of the greatest challenges when implementing a Genetic Algorithm is the coding of the solution traits. Amongst the most used options is a binary string (which is also the simplest to explain), where each location represents a certain characteristic that may (1) or may not (0) be present. The reason for this choice is that, with a binary string, crossover and mutation are very straightforward. For example, as shown in Table 3-1, a simple crossover option would be to have a constant (or random) cut point along the chromosome, and each offspring receives opposing parts of chromosomes from the parents.

Parent 1	110111 | 00110111000
Parent 2	010100 | 10010011001
Offspring 1	110111 | 10010011001
Offspring 2	010100 | 00110111000

Table 3-1 Crossover example - based on (Obitko 1998)

Mutation is also very straightforward, consisting of turning one gene to its opposite, 1 to 0 or 0 to 1.

Like Tabu Search, Genetic Algorithms have been applied to a variety of problems in finance, design and scheduling amongst others.

3.3. Ant Colony Optimization

The concept of artificial Ant Colony Optimization (ACO) was first developed by Marco Dorigo in his PhD thesis (Dorigo 1992). The general reasoning behind the algorithm is that a colony of ants can always find the shortest path between the nest and the food source, despite the fact that each individual ant is blind. How do they do this?

Each ant sets out to find the food source individually and, if presented with alternative paths, will randomly choose one, leaving behind a pheromone trail. Assuming there are two paths available, some ants will choose one and others will choose the other, but the ones that chose the shortest path will walk along it more frequently (more back and forth routes from nest to food in the same time frame), increasing the pheromone concentration there and more effectively countering its evaporation. Since ants will tend to follow the path with higher pheromone concentration, eventually the entire colony will converge to the shortest path as shown in Figure 3-3.

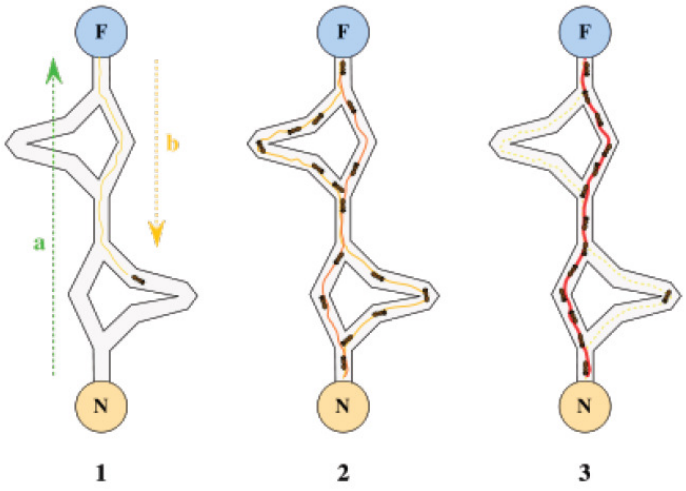

Figure 3-3 Colony converging on the shortest path – from (Shekhawat, Poddar and Boswal 2009)

Artificial ants need not be blind like the biological ones. In fact, we can use previous knowledge of the problem (sight) to guide the ants in the initial steps of the algorithm, before the pheromone trails have become dominant.

The application of this metaheuristics to problems such as the travelling salesman problem or the VRP is immediately recognizable. Usually, the probability that an ant will move from node *i* to node *j* will be given by:

$$p_{i,j} = \frac{\tau_{i,j}^{\alpha} \eta_{i,j}^{\beta}}{\sum \tau_{i,j}^{\alpha} \eta_{i,j}^{\beta}} \quad (6)$$

where τ is a matrix of pheromone and η is the desirability (or heuristic) matrix (the inverse of the distance for example). α and β are parameters that control the influence of each.

Whenever an ant chooses a path it will drop pheromones, increasing the corresponding value in the τ matrix by a factor relative to its performance (alternatively, only the best ants can be chosen to drop pheromone). At the end of colony's iteration the whole matrix suffers evaporation, which means that all the pheromone is reduced by a, usually fixed, percentage. This behavior allows the artificial ants to forget bad choices made in the past, and prevents the pheromone matrix from growing to infinity or saturating. Expression (7) represents the update of the pheromone matrix, with ρ being the evaporation coefficient.

$$\tau_{ij}(t) = (1 - \rho) \cdot \tau_{ij}(t - 1) \quad (7)$$

The end result will be the same as in the biological case. Better solutions will drop more pheromone, guiding the following iterations to better results

When originally presented, ACO was inferior to state-of-the-art heuristics and metaheuristics used to solve its first problem, the TSP. Regardless, it proved that the concept was good and encouraged further research.

Ant Colony Optimization as a metaheuristic was only proposed years later after the original application to TSP. Today, it is applied to extremely diversified fields such as assignments, routing and telecommunications (Stutzle 2005).

4. Multi-Depot Optimization

As was referred in Section 1.4, one of the goals of this work is to construct an application that is simultaneously easy to use and provides good results in solving the problem at hand. To that end, the application was divided into 3 parts: preprocessing, problem solving using various algorithms (optimization) and post processing schematically shown in Figure 4-1.

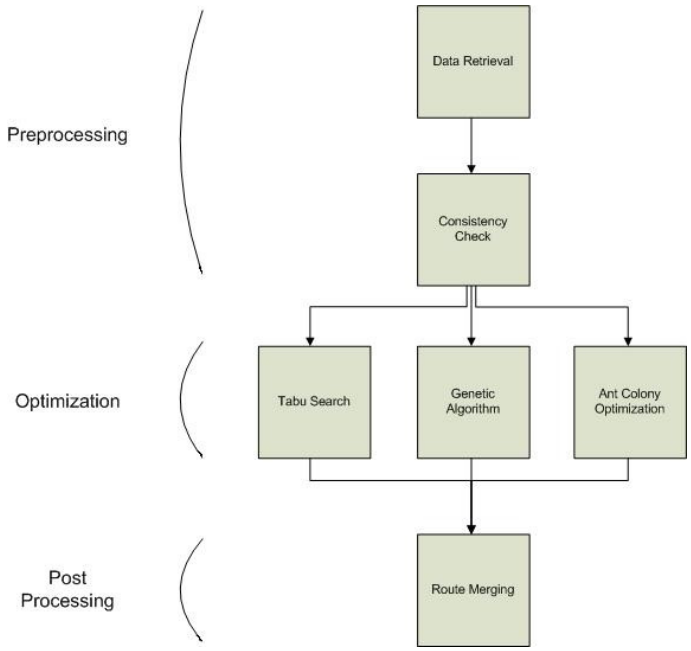

Figure 4-1 Implementation diagram

4.1. Preprocessing

The data retrieval for the application can be handled in three different ways. The user can supply them as a matrix of times taken between sites and plants, distances between them or two dimensional coordinates of each of them. The application assumes that time data is more precise than distance data, and that both are more precise than coordinate data. Therefore, if times are available they will be used, followed by distances and finally coordinates.

All of the relevant data are converted into time (if not already in that form) using the truck speed ($time = \frac{distance}{speed}$) and road non linearity coefficient (distance obtained from coordinates suffers a percentage increase) if required. Following this, a consistency check is performed on the data to answer two questions:

- Are any of the collection sites too far from all plants that the maximum travel time constraint would make them unreachable?
- Are there any collection sites supplying more than one truckload of mass?

If the answer to the first question is yes, then those sites are permanently removed from the problem data and a warning is displayed, informing the user of this action.

If the answer to the second question is yes, then those sites are divided into two new sites located in the same spot, with the first one supplying a quantity of mass equal to one full truckload and the second whatever is remaining. The process can occur multiple times for the same site if its supply is larger than 2 truckloads.

4.2. Implemented Algorithms

Three different algorithms were developed for solving the problem at hand, not only to provide the user with various alternatives but also to be able to run a scientific comparison between them. In the following subsections, the three algorithms are described in detail.

4.2.1. Tabu Search

The implemented algorithm was greatly based on (Renaud, Laporte and Boctor 1995). This was selected because, despite its age, it presents good results and could work as a comparison for the other algorithms that will follow.

4.2.1.1. Description

In order to provide an easier comprehension of the algorithm, the description is divided into 3 fundamental phases which will be described in turn.

- Clustering of collection sites around the closest plants

This first phase is a preparation for the optimization that will follow. Here the general multi-depot VRP is divided into N sub problems with just one plant, assigning each collection site to the nearest plant (there is no upper of lower limit for the number of sites assigned to each plant). The clustering must be done in a way that creates the sub-problems, organizes all the data relevant to them in new sub matrices, but still retains the original information so the collection sites may be later re-inserted in the general problem.

- Solution initialization using the Clark Wright Savings Algorithm or Improved Petal Heuristic

In this phase the application can either use the Improved Petal Heuristic (as done in the (Renaud, Laporte and Boctor 1995)) to initialize the solution if the data is supplied in coordinates or alternatively use the Clark Wright Savings Algorithm (the reason for introducing this algorithm is explained in Section 4.2.1.2). Regardless, the solutions produced by both are considered acceptable.

Either of the algorithms described in Section 2.2 is applied to the data obtained after the clustering. At the end of this phase two matrices will be obtained. The first, designated **R**, will contain general data for each route (associated plant, total travel time and used capacity, number of sites visited, first and last site) while the second matrix, designated **S**, will contain the ordered sequence of collection sites to be visited in the route. This data structure is similar, and was inspired by, the one

used in (Battarra, Baldacci and Vigo 2007). Table 4-1 and Table 4-2 show example instances of the **R** and **S** matrices.

Plant	Travel Time	Load	#Sites Visited	1st Site	Last Site
1	35.98	55	3	6	9
1	48.70	60	5	2	5
1	23.12	18	2	1	10

Table 4-1 Matrix R example

S				
6	4	9	0	0
2	8	7	3	5
1	10	0	0	0

Table 4-2 Example of S matrix

As can be seen in the example, the application of either heuristic to all sub problems will result in a number of matrices equal to double the number of plants. These will have to be merged into only 2 matrices (one **R**, one **S**) observing the correspondence between the local ordering of each sub problem and the global one.

- Application of Tabu Search to improve upon the previously found solution

Before starting the Tabu Search algorithm, an initial pre-optimization is applied to the initial solution, which simply consists of attempting various sub-tour reversals as described in (Hillier and Lieberman 2001) to every route. If any reversal provides a lower cost than the current one while not violating constraints, it is immediately accepted. To attain a higher degree of computational efficiency, when the route is small in comparison to the maximum number of iterations chosen, the algorithm will attempt to test all possible exchanges and, if no improvement is found, it ends. If the route is large it will randomly find reversals to evaluate until the maximum number of iterations is reached.

Following this, the fast improvement phase starts, which takes up the majority of the time and computational effort in the Tabu Search implementation. It can be divided into three distinct, but simultaneously similar, steps.

- **Inter-Plant Exchanges**

In this first step exchanges of collection sites between routes belonging to different plants are tested. The first plant is randomly selected while the second can also be selected randomly, or be the one closest to the first, according to a designated probability. Following this, a route is randomly chosen from each plant and then a collection site in each of these, randomly as well[6].

Having selected the plants, routes and sites, the following 6 movements are applied:

A. Swap sites between the two routes.
B. Insert first route's site in the second.
C. Insert second route's site in the first.
D. Insert two consecutive sites from the first route into the second.
E. Insert two consecutive sites from the second route into the first.
F. Swap two consecutive sites between routes.

[6] In the movements that follow, when more than one collection site is required or when it is necessary to determine if a site goes before or after another, this process is done randomly.

All movements use as a reference (the routes on which movements will be applied) the ones from the previous solution, but what happens after each movement is dependent on the result itself. Five different situations can happen, for which there are 3 possible actions.

a) Movement broke constraints.
b) Movement did not break constraints but its performance is inferior to the reference's and is either tabu or also inferior to the current performance[7] (or both).
c) Movement did not break constraints, its performance is inferior to the reference's but is superior to the current performance and is not tabu.
d) Movement did not break constraints and its performance is superior to the reference's and to the current performance.
e) Movement did not break constraints and its performance is superior to the best found until then.

Cases a) and b) suffer the same fate, they are discarded and both the reference and the current performance are kept unaltered.

For cases c) and d) the reference is maintained but the result is kept as the current solution, along with its current performance. This allows for the aspiration criterion to be present, as the tabu status is ignored if the solution is made better – case d) and simultaneously allows the acceptance of worse solutions to escape local minima, as long as they are not tabu – case c).

In the last case, e), the solution is immediately accepted and the movement cycle is broken regardless of which movement obtained the solution. The following iteration is started with this solution being used as the reference and kept as the best solution so far.

Whenever the sequence of movements ends and as long as the current sequence was changed at any point along it (or if the sequence was prematurely ended due to case e)), a local optimization similar to the pre-optimization described above is applied only to the routes that were changed.

- **Intra-Plant Exchanges**

The second step of fast improvement consists of exchanging collection sites between two routes belonging to the same plant. These are picked randomly, along with the sites inside each. At the start of this step the reference is updated to the current solution found in the inter-plant step, if any was found.

The movements for this step are identical to the ones described above, and so are the consequences for the results, with the obvious difference that they are now occurring between routes belonging to the same plant.

- **Three Route Exchanges**

The third and last step of fast improvement consists in exchanging collection sites between three different routes (regardless of which plant they belong to) and contains only one move, differing from the six movements of the previous steps. The three routes are selected randomly.

The movement consists of taking a site from the first route and inserting it in the second, then taking a different site from the second route and inserting it in the third. The actions following the result of said move are the same as the above.

[7] The current performance corresponds to the performance of the previous movement, or in the case of the first movement, to an artificially inflated value for easy updating

Naturally, whenever a movement is made it is included in the tabu list whose size is constant. When the maximum size is reached the older movements are progressively forgotten and replaced by new ones

To ease comprehension of the algorithm, which is relatively complex, a schematic of its different phases and steps has been constructed and is presented in Figure 4-2.

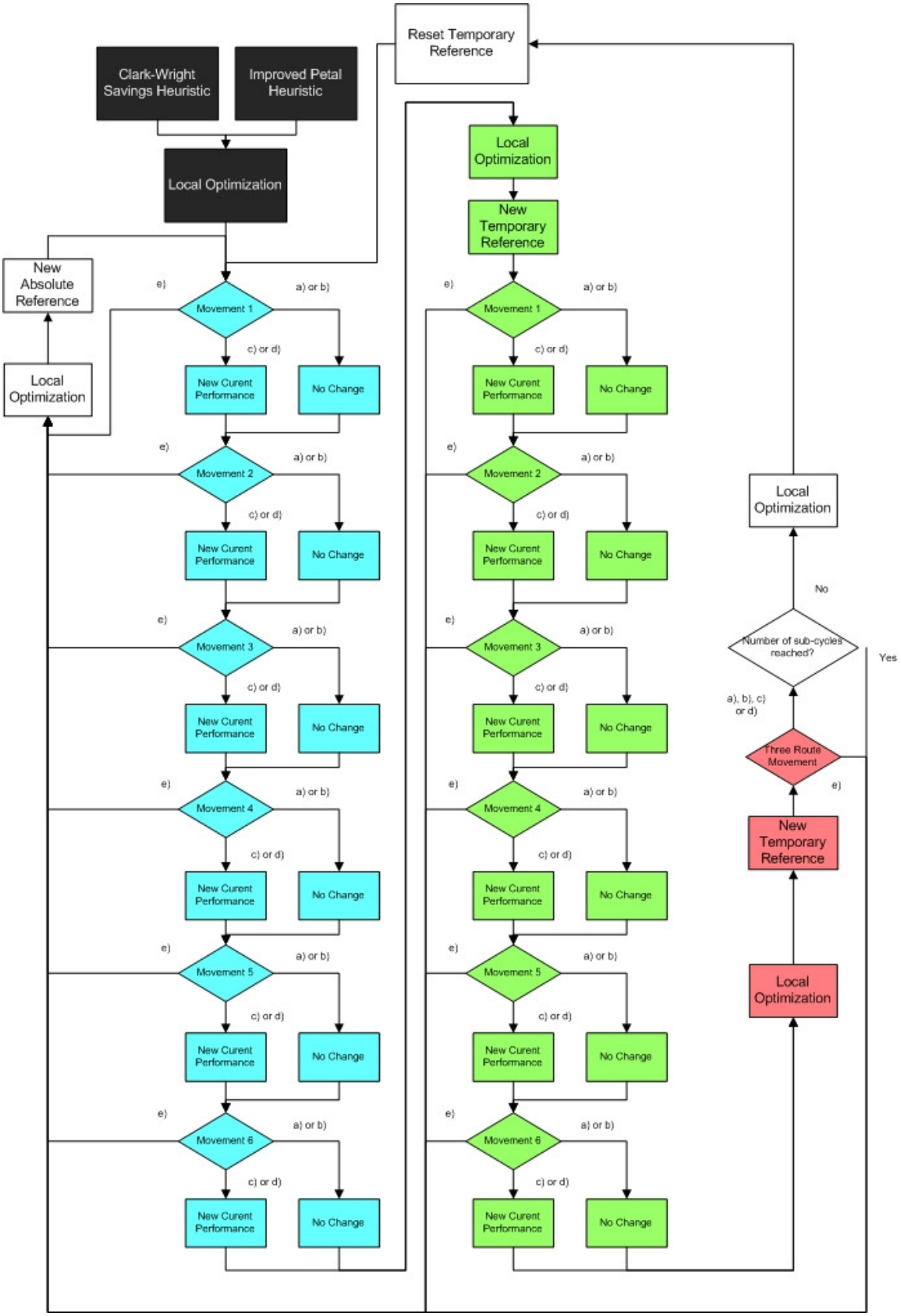

Figure 4-2 Fast improvement diagram

After fast improvement has finished, intensification begins. It is very similar to the second step of the fast improvement, the intra-plant exchanges. The difference lies in the fact that here two routes are chosen and the movements described above are applied repeatedly in an attempt to obtain improvement. Once the stopping criterion has been met, the algorithm moves to the next set of two routes and tries again.

The original implementation by (Renaud, Laporte and Boctor 1995) had an additional phase named diversification, which focused on inter-plant exchanges. This phase though, as referred by the authors themselves, was very computational intensive and only improved results on a small number of cases. With that said, it was chosen to leave this phase out of the current implementation.

4.2.1.2. Contributions

As was referred to at the start of the description of the Tabu Search algorithm, it was greatly based on (Renaud, Laporte and Boctor 1995). Even so, some changes were made to the original implementation, which will be described in detail below.

- Finding closest plants

In the inter-plant exchange step, the algorithm is required to find the closest route to the one that was first chosen (observing the previously chosen plants), in order to (most of the time) not waste computational effort in exchanges that would not make sense. In the reference, this measure of closeness is obtained by calculating the centroid of the locations that make up each route. Regretfully or not, the application accepts data as coordinates, distances and times so obtaining the centroid would be impossible in the two last cases. Due to this, a methodology was implemented in which the second plant is chosen as the closest (or a random one depending on the parameter probability) to the first and then a random route is picked in it.

- Introduction of the Clark Wright Savings Algorithm

For the same reason as above, data not always being supplied in coordinates, the Clark Wright Savings Algorithm was implemented in alternative to the improved petal approach. The Clark Wright Savings Heuristic was chosen due to its simplicity and acceptable results.

- Final local optimization

At the end of the algorithm (after intensification) an additional step of local optimization is applied to the whole solution but with a larger number of maximum iterations than before. This step, although probably redundant in most problems, may achieve small improvements on some at a negligible computational cost.

- Phased optimization

The last alteration is probably the one that will have a larger impact on the results. It was implemented to solve an odd behavior presented by the algorithm during testing. The problem consisted in the solution diverging consistently, in other words, starting from the initial heuristic solution, the following ones were worsened consistently until stabilizing around a bad solution. This resulted in no improvement at all at the end of the Tabu Search as can be seen in Table 4-3 and Figure 4-3.

Tabu Size	Probability	Solution after Fast Improvement	Improved Initial Solution?
25	0,3	639,79	No
25	0,9	639,79	No
50	0,6	639,79	No
100	0,3	639,79	No
100	0,9	639,79	No

Table 4-3 Results obtained on selected tests using no sub-cycles. Testing conditions equal to ones in Section 5.1

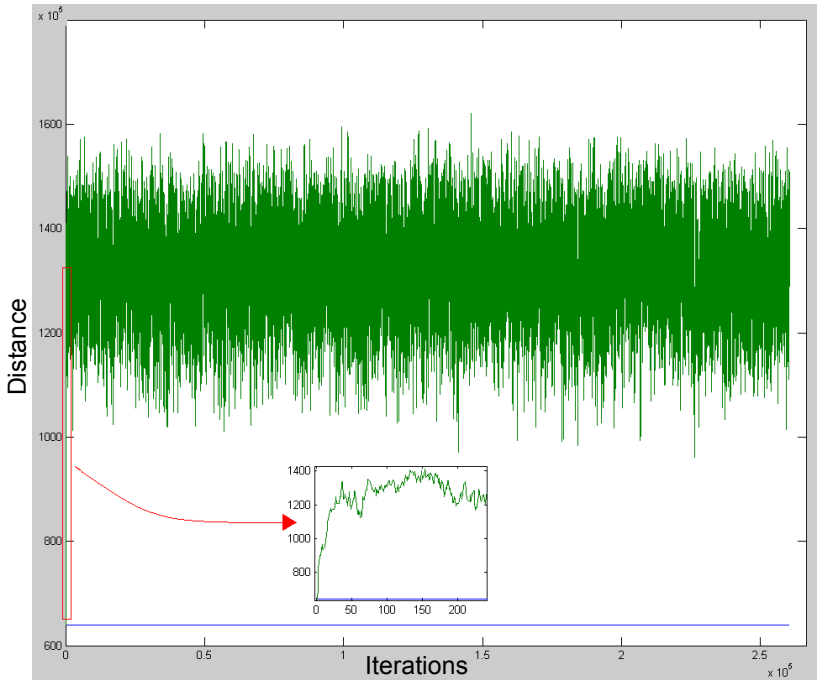

Figure 4-3 Demonstration of algorithm behavior with no sub-cycles (blue – best found, green – current solution)

The theory proposed for this behavior is that, due to the MDVRP problem being tightly constrained, the algorithm could go through all of the movements described above without finding a feasible solution. In other cases, it finds only one that is worse than the current but is still accepted under the pretext of escaping local minima. This being a frequent occurrence, the algorithm strays from the optimum and gets stuck in areas with poor performance.

The solution found was to introduce a new parameter, named Number of Sub-Cycles, which forces the algorithm to use the same reference a number of times before replacing it, despite it being better or worse than the ones found through movements. In this way, there is a larger exploration of the neighborhood before it switches to a new area.

4.2.2. Genetic Algorithm

The Genetic Algorithm implemented was greatly based on (Ombuki-Berman and Hanshar 2008). It was chosen due to being relatively recent and because the reference itself attempts to face the scarcity of work done in the application of GAs to this, multi-depot, version of the vehicle routing problem.

4.2.2.1. *Description*

Before beginning to describe the way the algorithm works, it is useful to first explain the coding of the genes, in other words, how chromosomes are used to represent vehicle routes in a univocal way. In the proposed algorithm, each chromosome represents a solution, containing a number of rows equal to the number of plants. In each row, the ordered list of sites to visit is placed as shown in Figure 4-4. There is no distinction between various routes belonging to the same plant within the chromosome, so there needs to be a route scheduler whose job is to break up the sites into feasible routes.

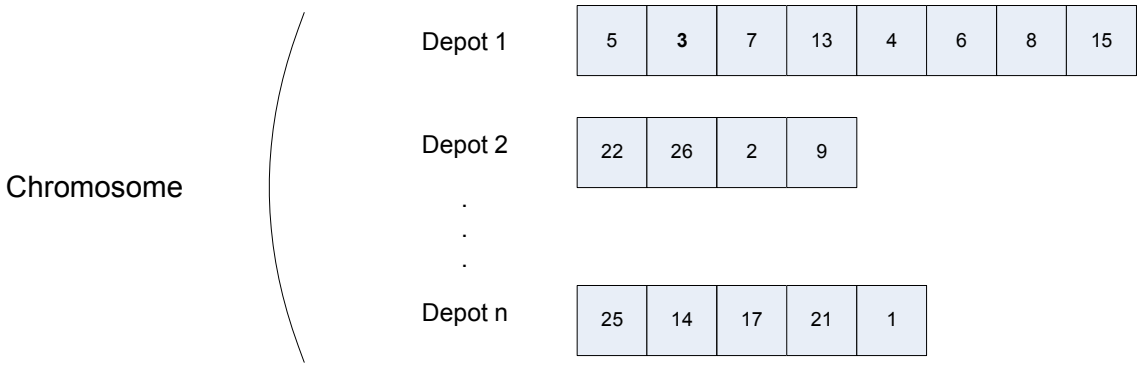

Figure 4-4 Chromosome example – based on (Ombuki-Berman and Hanshar 2008)

The algorithm can be divided into a pre-processing phase and an iterative phase.

- Pre-processing

The pre-processing consists of initially assigning collection sites to plants as well as creating the initial population. As for site assigning the process is similar to what was done for the Tabu Search in that we simply associate each site to its closest plant. In the Genetic Algorithm though, we also build a list of sites that can eventually suffer a plant change. As an example, one site might be closest to plant X but no far from plant Y as well. If this is the case, we assign it to plant X but remain conscious that it might also have been assigned to plant Y. A site is considered swappable when expression (8) is true, where *s* is the site, *p* the plant under consideration, c_p the closest plant and *Bound* a tunable parameter. This awareness, under the form of a swappable site list, will be important later when implementing mutation.

$$\frac{Distance(s,p) - Distance(s,c_p)}{Distance(s,c_p)} \leq Bound \qquad (8)$$

As for creating the initial population, it was done randomly in the (Ombuki-Berman and Hanshar 2008). The application mimics this behavior, but also optionally adds one chromosome that corresponds to the Clark Wright solution for the current problem. The solution must then be converted to chromosome form.

- Iterative phase

The iterative phase contains the majority of the algorithm. It starts with the decoding of the population's chromosomes using the route scheduler referred above. It works by starting a route with

the first site listed for each plant and then adding the following sites until no more can be added due to constraints. At this point, a new route is started and the process continues until the end of the plant. Once the routes are built, an analysis is done on the last site inserted in each route to see if the global cost would be lower if it was assigned as the first site of the next one. If so, the switch is implemented. The process is repeated for all plants.

With the routes built, obtaining each chromosome's fitness value is trivial, as it corresponds to the sums of costs of all routes that make it up. Crossover, or reproduction between chromosomes, can then begin, but not before the best, or a number of the best, chromosomes are selected and saved. These are elite chromosomes that will be cloned into the next generation.

Following this, a tournament selection strategy is used to find parents for the next generation. This consists of randomly selecting 2 chromosomes (the elite or elites can also be selected) and then choosing one of them by obtaining a random number. If it is smaller than 0.8 the best is selected, else any of them can be chosen. The same is done to obtain the 2^{nd} parent.

With both parents selected mating can begin. One plant is selected randomly, as only intra-plant mating occurs in this implementation. Then, one route is chosen from each chromosome, and the sites present in these routes are removed from the opposing chromosome. We then compute the reinsertion costs of sites and place them in the best locations found. Once all sites have been reinserted in both chromosomes they are placed in the new population and the process is repeated until we have enough descendants to make up a population with the same number of chromosomes as the previous one.

Mutation is the last step in the iteration. It can happen as intra-plant mutation and/or inter-plant mutation, and there is no guarantee that it will be present in all iterations. Any chromosome can be selected for mutation save the elite ones.

Intra-plant mutation can occur with a given probability, and happen in three different ways, each of them with equal probability (1/3) of happening. These are schematically shown in Figure 4-5.

a) Reversal mutation occurs when we select a plant belonging to a chromosome, select two cut points in said plant (regardless if they belong to the same route or if they cut through routes) and mirror the site ordering between the cut points.
b) Single customer re-routing consists of removing one site from anywhere in the chromosome and then computing all the re-insertion costs within the entire chromosome and placing it there. This step might end up being an inter-plant mutation after all.
c) Swapping will simply take 2 sites from the same plant and switch their locations.

Inter-plant mutation can occur independently of intra-plant mutation but is restricted to one type of move only. It uses the list containing swappable sites constructed at the start of the algorithm, by selecting one site from this list, removing it from its current location, and then randomly choosing a different plant to insert it in. Inside this plant's routes the location with the smallest reinsertion cost is selected and the site is placed there.

After using the mutation operators, if probability favored any of them, the next iteration begins using the new population created.

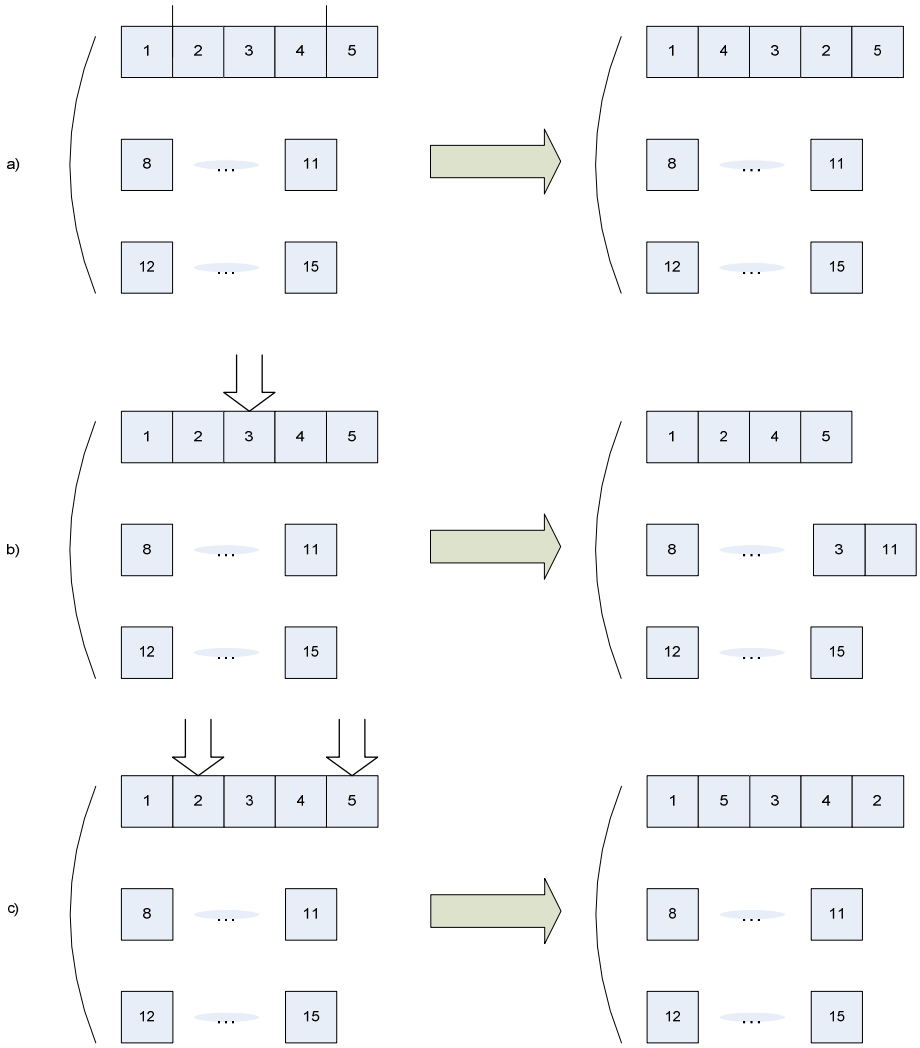

Figure 4-5 Intra-plant swapping examples

4.2.2.2. Contributions

Only a few changes were introduced in the original algorithm found in (Ombuki-Berman and Hanshar 2008).

- Route scheduler

The route scheduler present in the (Ombuki-Berman and Hanshar 2008), after constructing all the plant's routes, proceeds to analyze the possibility of switching the last customer of each route to the next, starting with the first one.

In our implementation, this 2nd phase of the route scheduler starts with the last route. For example, in a 4 route setting, our first attempt will be to reroute the last site of the 3rd route to become the first of the 4th, then the last of the 2nd to become the first of the 3rd and so on.

The reason for this change is that, since the first phase of the route scheduler fills routes until a constraint is met, this will usually mean that all but the last route will be tightly constrained, so

switching sites between them will probably be unfeasible. If we start with the last route, we can keep making room for sites in the previous ones if switches are successful.

- Crossover feasibility

In the reference, during crossover, the algorithm checks for feasibility when analyzing the reinsertion of sites back into the chromosome. In the application, this feasibility check was disregarded as, due to the nature of the chromosome representation all routes are feasible. The path taken was to apply the route scheduler to all the insertion possibilities and analyze the total cost of the solution, regardless of it breaking up existing routes and/or creating new ones.

4.2.3. Ant Colony Optimization

Due to the lack of work in applying Ant Colony Optimization specifically for the Multi-Depot Vehicle Routing Problem, a new algorithm was implemented. Even though it has no specific source, it was inspired by (Silva 2005) and also by the Genetic Algorithm application cited in the previous algorithm (Ombuki-Berman and Hanshar 2008). Various other sources were also consulted such as (Bella and McMullen 2004) and (Dong and Xiang 2006).

4.2.3.1. Description

Much like the previous algorithms, a preprocessing phase precedes the actual improvement phase.

- Pre-processing

The algorithm begins with a pre-processing phase that is exactly the same as the one used for the implemented Genetic Algorithm. All the collection sites are clustered around a single plant but a list is also constructed of which sites may eventually be assigned to which plants. As will be seen later, in this implementation the original clustering is meaningless and only the possibilities presented in the above list will be relevant to the progression.

Following the clustering, the pheromone and heuristic matrices are initialized. For the pheromones, the matrix is simply set to 0.5 for all possible connections between sites and plants. The heuristic matrix on the other hand is initialized according to a modified savings heuristics (Silva 2005):

$$\eta(i,j) = d(i,1) + d(j,1) - 2.d(i,j) + 2.|d(i,1) + d(j,1)| \quad (9)$$

where η is the heuristic matrix, d is the matrix of time taken between locations, and i and j represent two sites or one site and one plant. Following this, the heuristic matrix is normalized so that its values are between 0 and 1.

We are now ready to proceed with the iterative phase of the algorithm.

- Iterative phase

The iterative phase starts with one ant being placed at each plant. Then we start by looking at the first ant (located at the first plant) and computing the attractiveness[8] of every remaining site that it might travel to (using the list of possible sites assigned to its plant). We then make it travel to one of

[8] The attractiveness is calculated according to equation (6) in Section 3.

the sites, with the chance of picking a site being related to its attractiveness. The process is repeated for every ant that starts at all the other remaining plants.

Now that all ants have moved once, the algorithm continues to move the ants in the same fashion. Every time a site is visited it is added to a tabu list so that no other ant can travel there again. Also, if at any point an ant can no longer move to a new site due to capacity or time constraints, it is routed back to the plant and restarts its movement from there. The process goes on until all sites have been visited once. Below, Figures 4-6 to 4-9 represent a possible movement pattern for a problem with 3 plants and 15 collection sites.

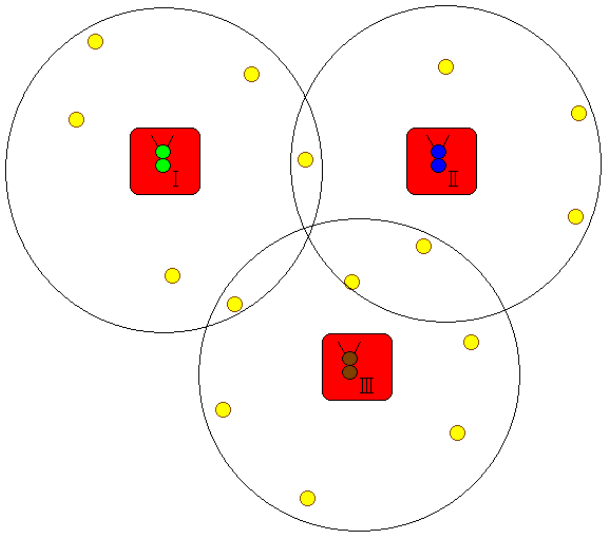

Figure 4-6 ACO algorithm – Stage 1: Ants starting at plants

In Figure 4-6 three ants (I, II and III) can be seen at their respective plants (red rounded squares) at the start of the iteration. Around them, the 15 collection sites (yellow circles) are distributed and the large black circumferences represent the collection sites that are accessible to each plant (through the list constructed during clustering).

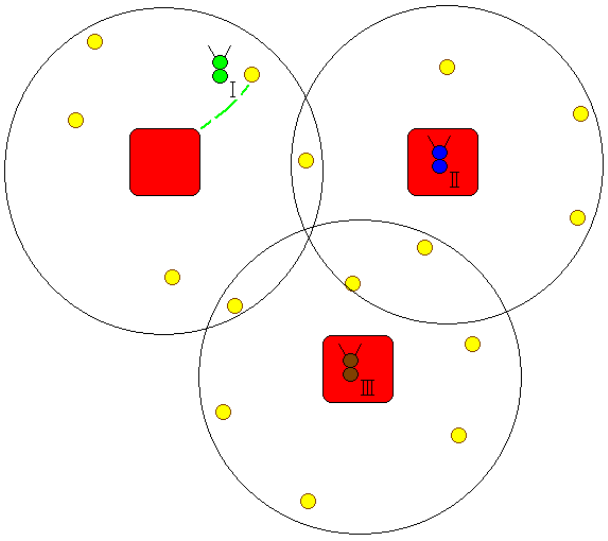

Figure 4-7 ACO algorithm – Stage 2: Ant I has moved

Figure 4-7 shows that ant I was the first to move and has chosen to go northeast. Since this is the first iteration, its decision is based solely on the heuristic information matrix as the pheromone matrix is equal for all routes. Regardless, and because all decisions are probabilistic, a different collection site might have been selected.

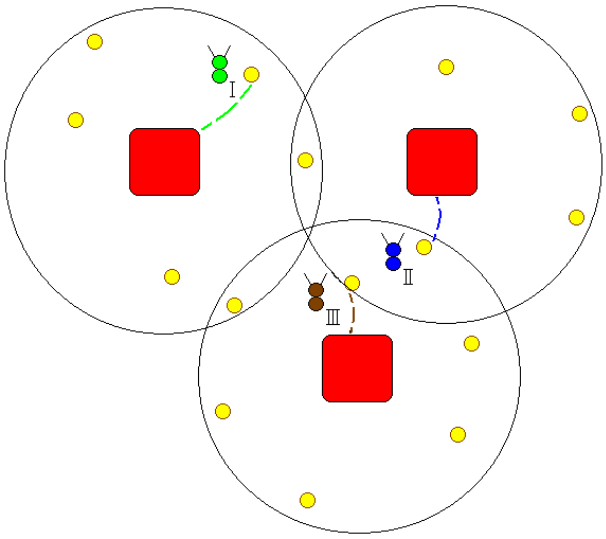

Figure 4-8 ACO algorithm – Stage 3: All ants have moved once

In stage 3, the remaining ants have also made their move (even though in the actual algorithm, they would move one at a time). As can be seen in Figure 4-8, both ant II and ant III have decided to move into a region that could be used by both. This means that, despite the original possibility of being assigned to ant II, the site just visited by ant III now belongs to its plant until the end of this colony's tours. The same is true for ant III and the site just visited by ant II.

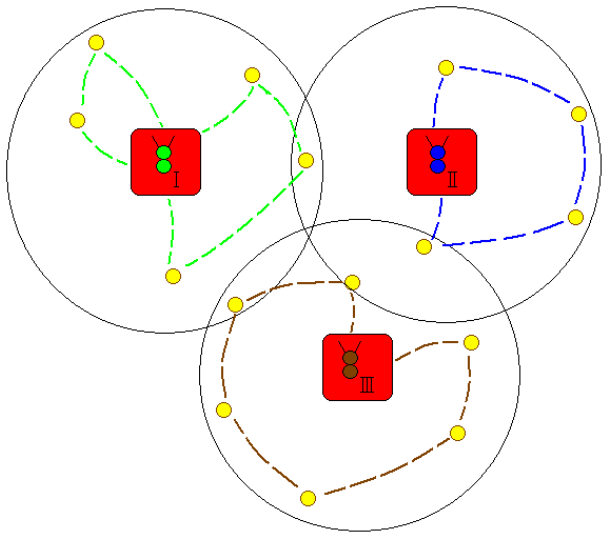

Figure 4-9 ACO algorithm – Stage 4: End of this colony's attempt

In Figure 4-9 all ants have completed their movements and all collection sites have been assigned a spot in one tour. We can also see that, in this case, ant I had to return to the plant during its tour because it had reached one or more constraints. This terminates the colony's attempt at the problem. Its total performance is stored and a new run is started (which can be interpreted as a

different colony having a hand at the problem) until the number of colonies used per iteration has been reached.

The reasoning behind the algorithm is that, since many collection sites can be assigned to more than one plant, the first ant to get to it will take it for itself, not allowing it to be visited by ants from other plants. This constitutes a controlled random behavior, because as different colonies attempt to solve the problem they will probably make use of different site/plant assignments. Due to this, the bound parameter is of the utmost importance in this algorithm as it will determine how far an ant can reach from its plant to visit a collection site. A slightly different form of this algorithm was attempted which will be explained in the contributions Section (4.2.3.2).

Finally, after all the colonies have attempted to solve the problem, their performance is compared and the best amongst them are selected to drop pheromone. The best colony of all will drop a fixed amount of pheromone on the paths its ants traveled, while the 2nd best (if more than one pheromone dropper as been selected) will drop a fraction of what the 1st dropped that is dependent on how many droppers are select, e.g., if we choose that 3 colonies will drop pheromone and the amount dropped is 0.3, the best colony will drop 0.3, the 2nd best will drop (2/3)*0.3 = 0.2 and the 3rd best will drop (1/3)*0.3 = 0.1.

Once pheromone dropping has been concluded, each matrix element suffers evaporation equal to a fraction of its value and the matrix is also saturated between 0 and 1. The iteration then concludes with the storing of the best route and the cycle repeats using the new pheromone matrix.

The following pseudo-code systematizes what was presented above:

Ant Colony Optimization algorithm

1. *Initialize iterations, i = 0.*
2. *Set i = i + 1. If i = maximum iterations terminate, else initialize colony counter, j = 0.*
3. *If j < (colonies used per iteration) proceed, else go to 10.*
4. *Set j = j + 1. Clear tabu list and place all ants at plants.*
5. *Set k = 1.*
6. *Compute attractiveness of all ants to all available sites. If no sites are available for any ants go to 9.*
7. *Move ant k according to heuristic, pheromone and probability and place site in tabu list. If no movement is possible due to constraint being reached, move to plant and reset capacity and length and repeat step. If no movement is possible because site list has been exhausted skip movement.*
8. *Set k = k + 1. If k <= (number of plants) go to 7, else go to 5.*
9. *Save current routes and performance and go to 3.*
10. *Compare performance of all colonies, save best and allow them to drop pheromone.*
11. *Evaporate pheromone from entire matrix and go back to 2.*

4.2.3.2. Contributions

Despite the references used for this section, this algorithm can be considered a contribution as a whole. This section will be used to explain an alternative algorithm that was attempted but was less successful than the one presented above. The only difference between both algorithms lies in the movement of the ants.

- Algorithm Flaw

As explained above, in the implemented version, the 1st ant will attempt to move to a new site, followed by the 2nd and so on, only returning to move the first again after all others have attempted their own move. A flaw that can be easily diagnosed is that this behavior tends to force the assignment of an equal number of collection sites to all plants which can be very suboptimal as can be seen in Figure 4-10 and Figure 4-11.

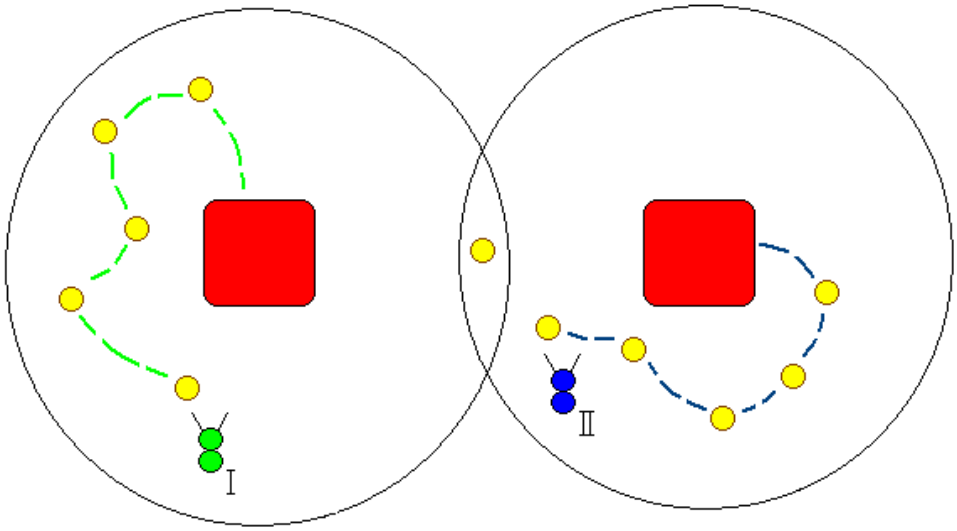

Figure 4-10 Ants after 5 movements each

Figure 4-10 shows a problem with only two plants. Both ants have now moved 5 times and there is only one collection site left to be assigned. Intuitively, it can be plainly seen that moving ant II to it would be a much better option, but as it has just moved it is now ant I's turn.

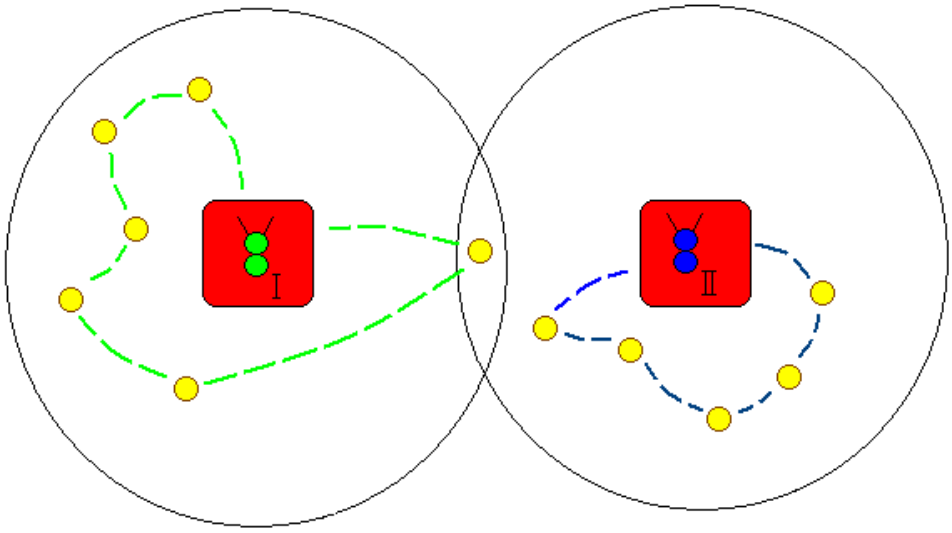

Figure 4-11 Final tours

The resulting tours shown in Figure 4-11 are suboptimal, and could be easily improved.

- Alternative Algorithm

The alternative algorithm first calculates the attractiveness of all nodes for all ants. Then, still using probabilities, it chooses simultaneously which ant to move and where to move it. It would almost certainly have offered a much better solution to the above situation as ant I would have a very low probability of moving to the remaining site, compared to the probability ant II would have.

In theory, the results should be better for most problems. What happens in practice though is that the additional calculations that must be made, computing the attractiveness of every node for every ant before any movement is done, are a heavy computational burden. The number of total iterations done for the same amount of time is largely inferior, and this difference is larger as we increase the number of plants.

4.3. Post Processing

Regardless of the algorithm used for optimization, the end result is always two matrices similar to the ones presented in (Table 4-1 Matrix **R** example) and (Table 4-2 Example of **S** matrix), containing all the solution information.

We could directly export this information, but what happens in many cases is that we have more than one route for each plant. These routes are usually constrained by either the truck's capacity or the route's own length. If length is the active constraint, then not a lot can be done, and the routes will have to be performed by different vehicles, probably with different drivers. But if the active constraint is the vehicle capacity, then the truck can simply return to the plant, unload, and start another route.

It is then worthwhile to check if any number of routes from each plant can be feasibly merged to form one, larger route that stops at the plant for unloading. This constitutes the famous bin packing problem.

The bin packing problem consists of having an infinite number of bins with a fixed capacity in which a certain number of items, each with its own size or weight, must be inserted. The objective is to fit all items in the smallest number of bins possible (Pinedo 2005) as shown in Figure 4-12.

.

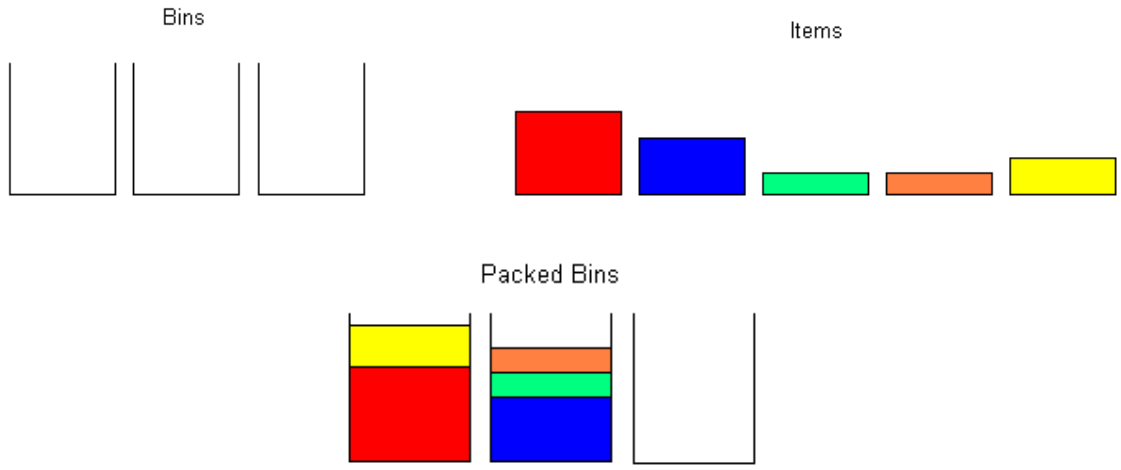

Figure 4-12 Sample bin packing problem and possible solution

In the present case, the bins correspond to the maximum vehicle travelling time and the items to the routes found during optimization. The more empty bins we can find, the less vehicles will be required to service all the routes.

To solve the bin packing problem the First Fit Decreasing (FFD) Heuristic will be used. It is simple, fast and yet has a good degree of accuracy. It consists of first ordering the items (routes) in order of decreasing size (total length). Then, each route is inserted into the 1st bin it fits into (Pinedo 2005).

The finalized routes are ready to be exported and displayed to the user.

5. Results

This section will present the results obtained while testing the various optimization algorithms. The first subsection will refer to the testing done to obtain the best set of parameters for each algorithm, followed by performance testing on various benchmark instances, and also on one new problem that describes what the application might find when applied to real life situations.

All testing was done using a computer with an Intel Core 2 Quad Processor Q8200 CPU at 2.33 GHz and with 3,25 GB of RAM, running on Windows XP and using MATLAB R2007a.

5.1. Parameter Tuning

To tune the parameters of each algorithm, tests were done on the p01 instance (Díaz 2007) which is a problem containing 50 collection sites and 4 plants for which the best know solution is 576,87 (measured in distance). A single test will be done for each set of parameters and it should last approximately 30 minutes. All measures of performance are given in percentage error relative to best know solution ($Error = \frac{Solution - BestKnow}{BestKnow} * 100$). In this section mostly graphical analysis will be presented, the full information in table format can be found in Annex B.

5.1.1. Tabu Search

For the Tabu Search algorithm described in Section 4.2.1, there are three parameters that might affect the solution quality (excluding the stopping criteria) which will be described below:

- ***Tabu size***

  This parameter will affect the balance between the attempt to escape local minima and the search for better solutions in the neighborhood. The larger the size of the tabu list, the less computational effort will be wasted repeating movements that have already been tested. However, some promising solution areas might not be explored as thoroughly as necessary.

  Tabu sizes of 25, 50, 75 and 100 were tested, corresponding to half, one, one and a half and two times the number of collection sites.

- ***Probability of choosing closest plant***

  As stated in previous sections, in the inter-plant exchange phase there is a parameter that controls the probability of the second plant being the closest to the first or a random one. Again, this influences the degree of exploration of the algorithm as higher values of

probability diminish the chance of senseless exchanges, but may also reduce the odds of finding good, less obvious, ones.
Probabilities of 0,3, 0,6 and 0,9 will be tested.

- ***Number of Sub-Cycles***

  Lots of sub-cycles will allow the algorithm to better explore the neighborhood of the current solution, but an exceedingly high number might cause it to waste too much time in each place, when the better areas might be far off. Values of 5, 20, 35 and 50 will be tested.

A total of 96 tests (48 using Clark Wright initialization and another 48 using the Improved Petal heuristic) will be done.

The testing conditions will be the following:

1. Maximum number of iterations of local optimization: 500
2. Maximum number of iterations of fast improvement: 1.000.000.000
3. Maximum number of iterations without improvement of fast improvement: 1.000.000.000
4. Maximum time spent in fast improvement: 1740s (29 minutes)
5. Maximum number of iterations of intensification per route pair: 100
6. Maximum number of iterations without improvement of intensification per route pair: 50

The reason for the high value of parameters 2 and 3 is so that we can guarantee that the stopping criteria is the computational time, which will allows us to compare results of testing with varying number of sub-cycles (this parameter influences the total number of iterations greatly). 1740 seconds corresponds to 29 minutes, leaving at the most one minute for intensification.

5.1.1.1. Clark Wright Results

In Figures 5-1 to 5-4 a graphical analysis is presented regarding the results obtained while testing Tabu Search with Clark Wright initialization.

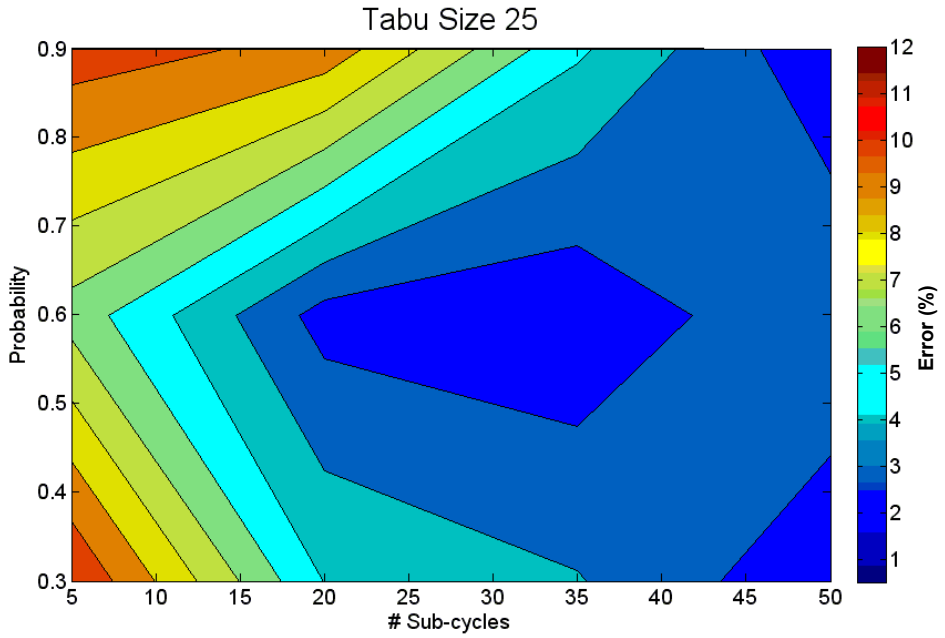

Figure 5-1 Tabu Search with Clark Wright initialization: Tabu size 25

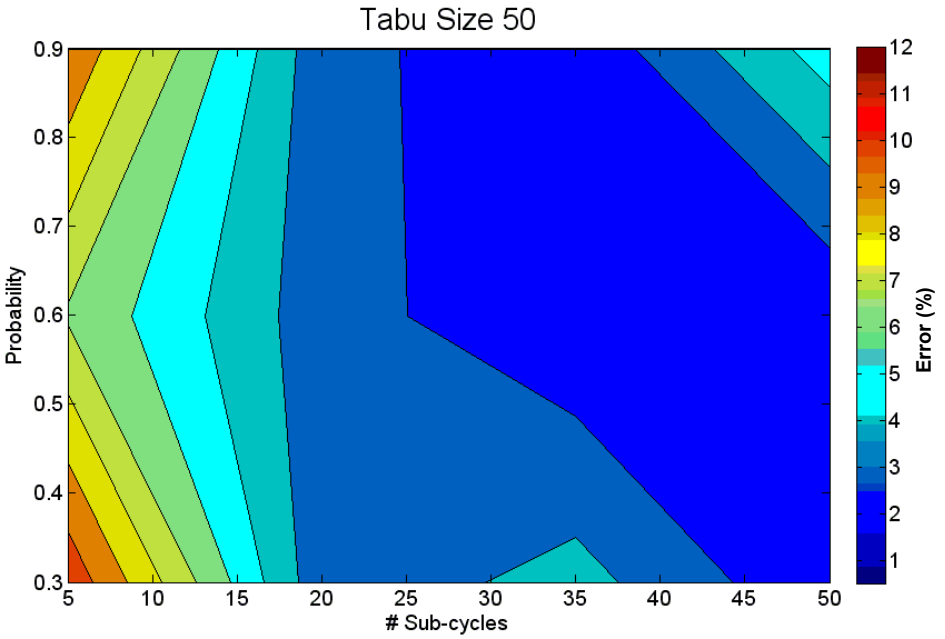

Figure 5-2 Tabu Search with Clark Wright initialization: Tabu size 50

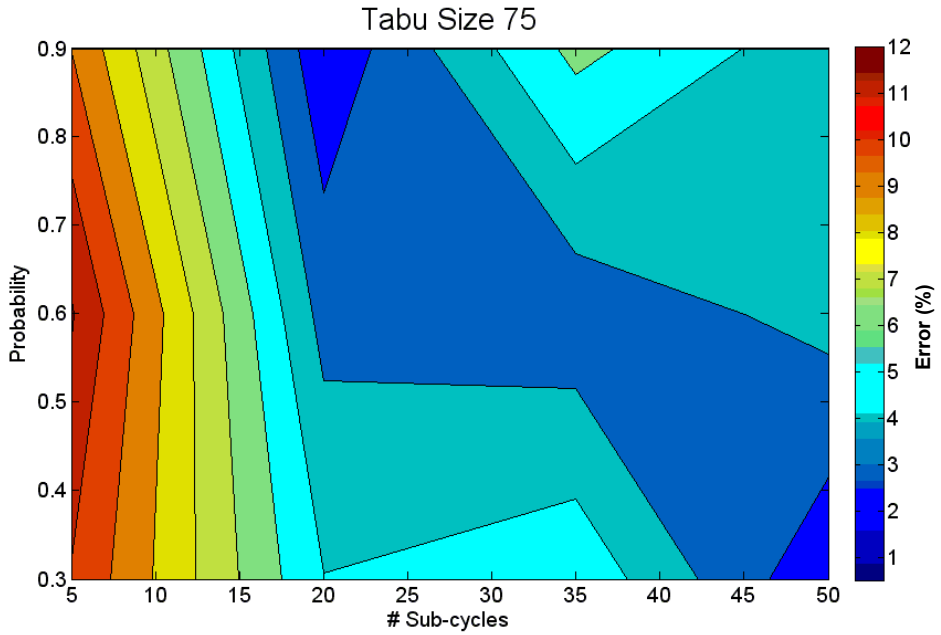

Figure 5-3 Tabu Search with Clark Wright initialization: Tabu size 75

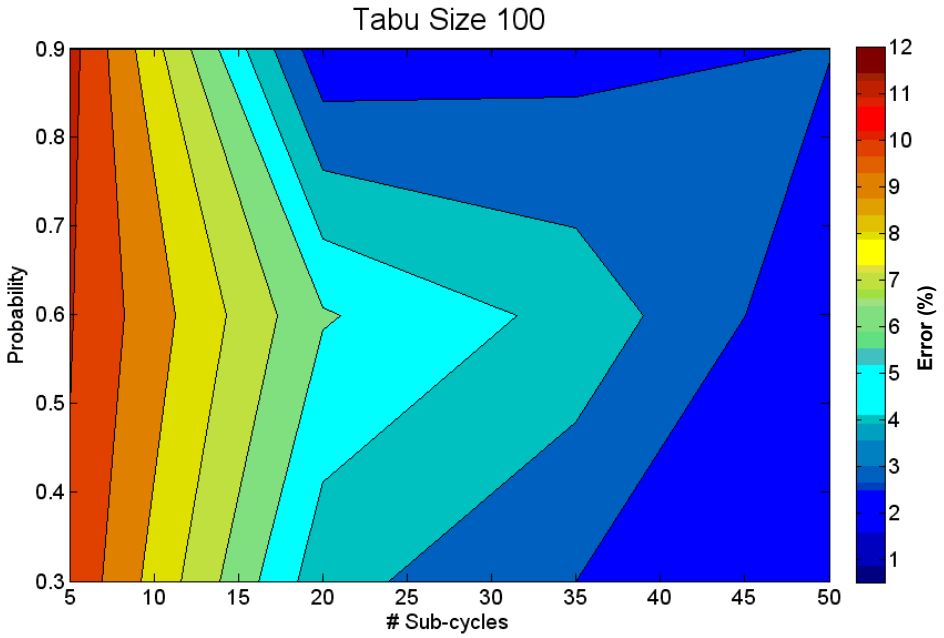

Figure 5-4 Tabu Search with Clark Wright initialization: Tabu size 100

In all of the figures, it can be readily observed that having a number of sub-cycles under 20 wields very poor results regardless of probability and tabu size. This reinforces what was stated in Section 4.2.1.2 that sub-cycles are essential to finding good solutions.

Observing the first two figures, the best results seem to be in the middle right side, which is to say relatively high number of sub-cycles (35 or 50) and a probability of 0.6. The third figure, showing tabu size 75, is hard to interpret as there are two distinct and small regions of good results and the fourth shows the best results at either low probability/high number of cycles or high probability/any number of cycles (save for a cycle number under 20 as was stated above).

As for a comparison between various tabu sizes, since it is hard to discern which is best through the figures, Table 5-1 was constructed listing the average error of each test at the respective tabu size and the average error of the three best tests of each. These averages were constructed using all the tests done at the corresponding tabu size[9]:

Tabu Size	Average Error (%)	Average Error of Best 3 (%)
25	4,11	2,19
50	**3,19**	**2,17**
75	4,12	2,58
100	3,37	2,19

Table 5-1 Performance at various tabu sizes for Clark Wright initialization

At tabu sizes of 50 and 100 the algorithm performs better, with a slight advantage for 50. It is assumed that tabu size, number of sub-cycles and local optimization scales with problem size (number of collection sites) while probability remains constant.

In light of what was said, the optimal parameters chosen for Tabu Search with Clark Wright initialization are the following:

- Local Optimization Maximum Number of Iterations: 10*Number of collection sites
- Tabu Size: Number of collection sites
- Probability: 0.6
- Number of Sub-cycles: Number of collection sites

5.1.1.2. Improved Petal Results

In Figures 5-5 to 5-8 a graphical analysis is presented regarding the results obtained while testing Tabu Search with improved petal initialization.

[9] In the elaboration of this table the tests with 5 sub-cycles were not considered as they have been proved to be sub-optimal.

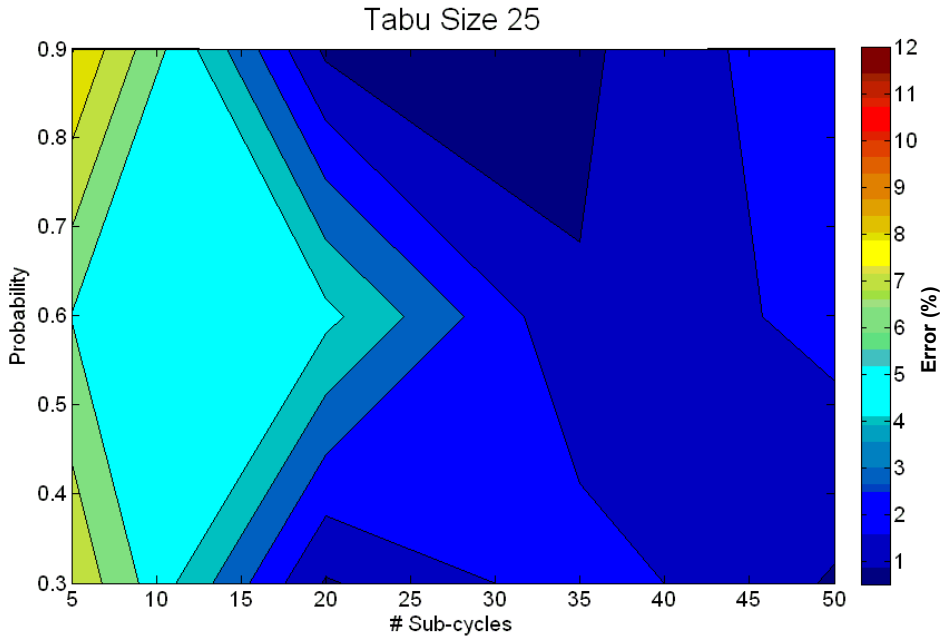

Figure 5-5 Tabu Search with improved petal initialization: Tabu size 25

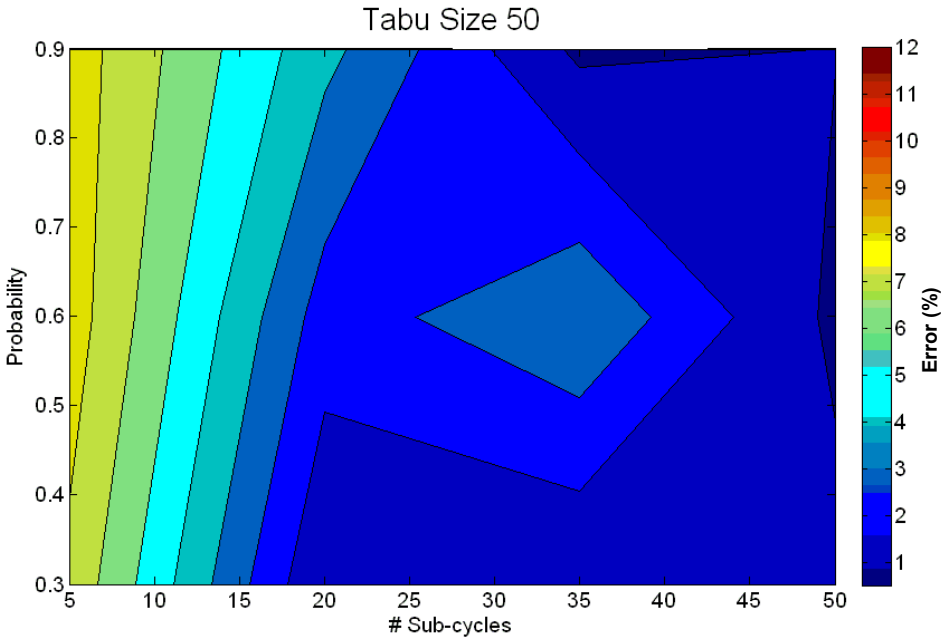

Figure 5-6 Tabu Search with improved petal initialization: Tabu size 50

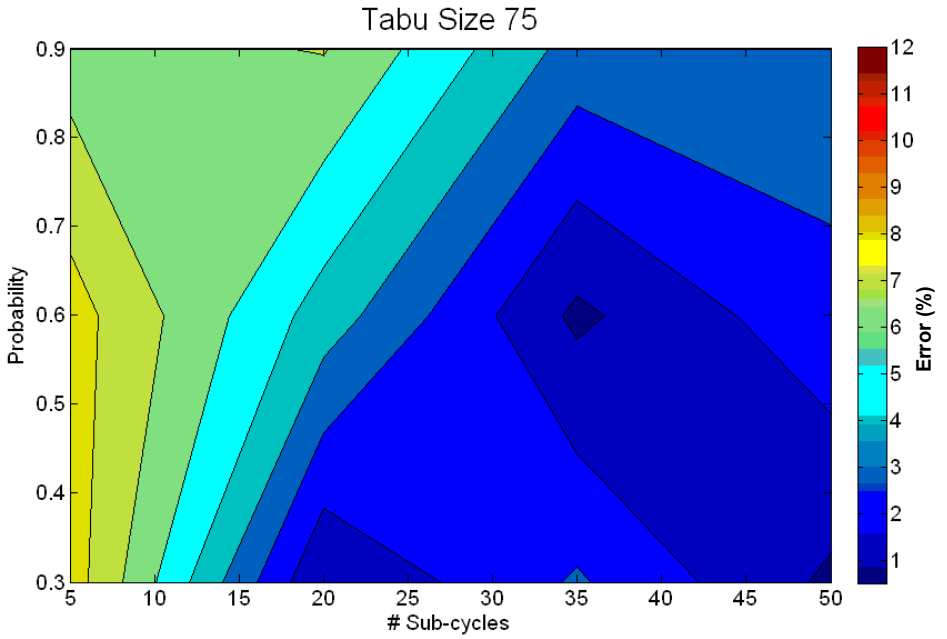

Figure 5-7 Tabu Search with improved petal initialization: Tabu size 75

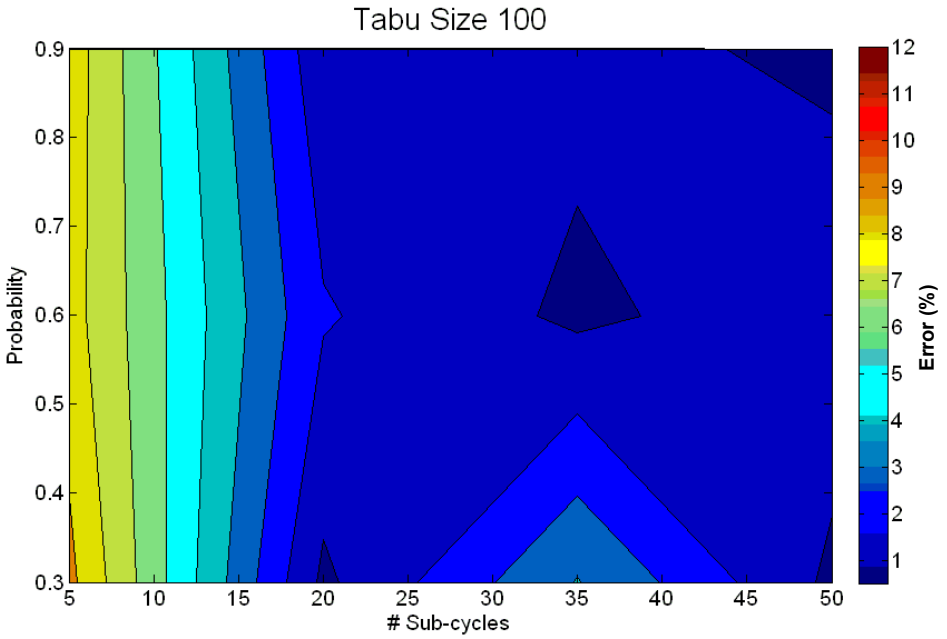

Figure 5-8 Tabu Search with improved petal initialization: Tabu size 100

Regarding numbers of sub-cycles under 20, the situation described for Clark Wright initialization is present again. The interaction between probability and tabu size is reversed though.

Observing the first two figures, the best results seem to be in at the top, which is to say a probability of 0.9 and a moderate to high number of sub-cycles (35 or 50). The third and forth figures show that the results at probability 0.6 and 35 sub-cycles prove to be best.

Table 5-2 is similar to the one presented above is now constructed for Improved Petal initialization in order to better compare tabu sizes to each other:

Tabu Size	Average Error (%)	Average Error of Best 3 (%)
25	1,95	0,82
50	1,85	0,87
75	3,02	0,89
100	**1,50**	**0,79**

Table 5-2 Performance at various tabu sizes for Improved Petal initialization

This time, a tabu size of 100 dominates all others. It is again assumed that tabu size, number of sub-cycles and local optimization scales with problem size (number of collection sites) while probability remains constant.

In light of what was said, the optimal parameters chosen for Tabu Search with improved petal initialization are the following:

- Local Optimization Maximum Number of Iterations: 10*Number of collection sites
- Tabu Size: 2*Number of collection sites
- Probability: 0.6
- Number of Sub-cycles: Number of collection sites

5.1.1.3. Initialization Comparison

In order to compare the Clark Wright algorithm to the Improved Petal heuristic for the purposes of initializing this Tabu Search algorithm, one can refer to either the figures on pages 36 through 40 or the tables on pages 38 and 41. Observing the figures, there are a higher number of blue areas, as well as darker blue sections in the Improved Petal tests when compared Clark Wright ones. Using the tables, Improved Petal errors range from approximately one half to three thirds of Clark Wright ones.

This proves that in virtually every situation the Improved Petal heuristic will provide better results and should be used if possible. It has not been mentioned yet, but while Improved Petal wields a solution using 12 routes, Clark Wright can only obtain a minimum of 13 for the instance at hand. This factor is of great importance, as the Tabu Search algorithm retains the number of routes supplied by the initial solution.

With that said, Tabu Search will only use Clark Wright initialization if the data is not supplied in coordinates or if, for some reason, Improved Petal optimization fails.

5.1.2. Genetic Algorithm

For the Genetic Algorithm described in 4.2.2, there are four parameters that might affect the solution quality (excluding the stopping criteria) which will be described below:

- ***Population size***

  The size of the population is a parameter that must be tuned in nearly all applications involving Genetic Algorithms. Along with crossover rate and selection method, it establishes the balance between great exploration or faster convergence. Smaller populations lack diversity but save computational time by focusing on a smaller number of calculations per iteration. For our testing purposes, populations of 25, 50 and 100 chromosomes were tested.

- ***Inter-Plant Mutation Probability***

  This parameter will control how often inter-plant mutation is applied. In usual GAs, mutation probabilities are often very low (Busetti 2000) but in our case mutation (and this type of mutation specifically) is a very important part of the algorithm as it is the sole way of moving collection sites from one plant to another (bar one special case, see below). Therefore higher than usual percentages for mutation were tested, specifically 12,5%, 25% and 50%.

- ***Intra-Plant Mutation Probability***

  This parameter controls how often intra-plant mutation is applied. Intra-plant mutation will probably have a lower effect on the overall exploration of the algorithm as, save for the case of single customer rerouting, it will only operate inside the same plant, mimicking what reproduction already does. For this reason, lower percentages were tested, namely 10% and 20%.

- ***Bound***

  The bound parameter is used only in the pre-processing phase of the algorithm, and determines how close a given site must be to a plant in order for the possibility of associating them to be considered. The expression used to construct the list of plants that may be assigned to a site follows, where *s* is the site under consideration, *p* the plant under consideration and c_p is the closest plant:

$$\frac{Distance(s,p) - Distance(s,c_p)}{Distance(s,c_p)} \leq Bound \qquad (8)$$

  Values of 1 and 2 were tested for the bound parameter.

A total of 36 tests will be done with the following testing conditions:

1. Maximum number of iterations: 1.000.000.000
2. Maximum number of iterations without improvement: 1.000.000.000
3. Maximum time spent: 1800s
4. Elite percentage: 1%
5. Use Clark Wright initialization: Yes

The reason for the high value of parameters 1 and 2 is so that we can guarantee that the stopping criteria is the computational time, which will allow the comparison of results obtained while testing with different population sizes. 1800 seconds corresponds to 30 minutes, which is also approximately the time spent in Tabu optimization. The elite percentage corresponds to just one elite member per iteration since the population is equal to or smaller than 100.

Due to the difficulty in representing 4 varying parameters at the same time, the intra-plant mutation percentage was chosen as the parameter to be examined first. Table 5-3 summarizes the average performance of tests under different mutation conditions.

Population Size	Intra-Plant Mutation (%)	Average Error (%)
25	**10**	8,58
	20	**6,04**
50	**10**	7,91
	20	**6,34**
100	**10**	8,63
	20	**7,40**

Table 5-3 Performance at different intra-plant mutation percentages

As can be seen, the 20% mutation rate tests consistently outperform the 10% ones. Also, a direct comparison between tests where the only parameter change is this mutation rate shows that the 20% results are better in 15 out of 18 comparisons.

The intra-plant mutation percentage will be fixed at 20% and the remaining parameters will be plotted in Figures 5-9 to 5-11.

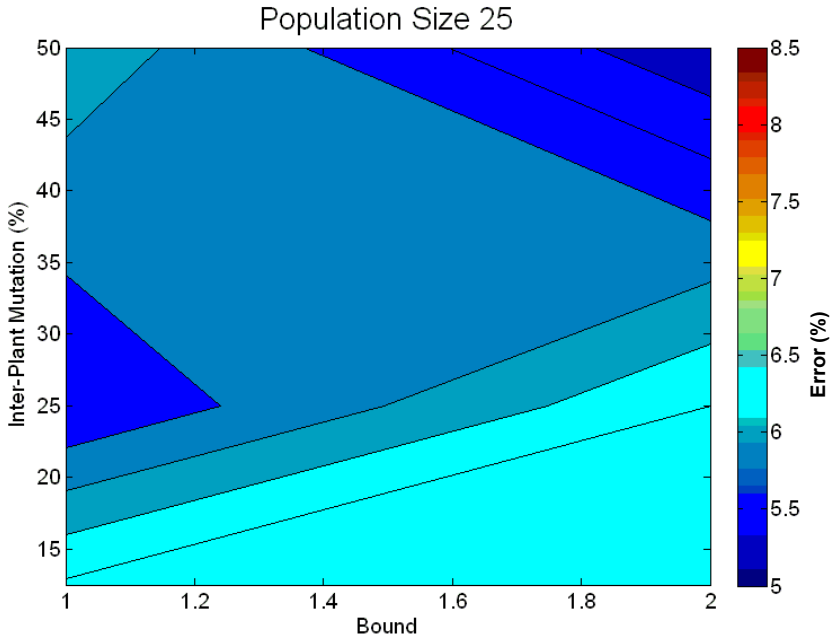

Figure 5-9 Genetic algorithm: Population Size 25

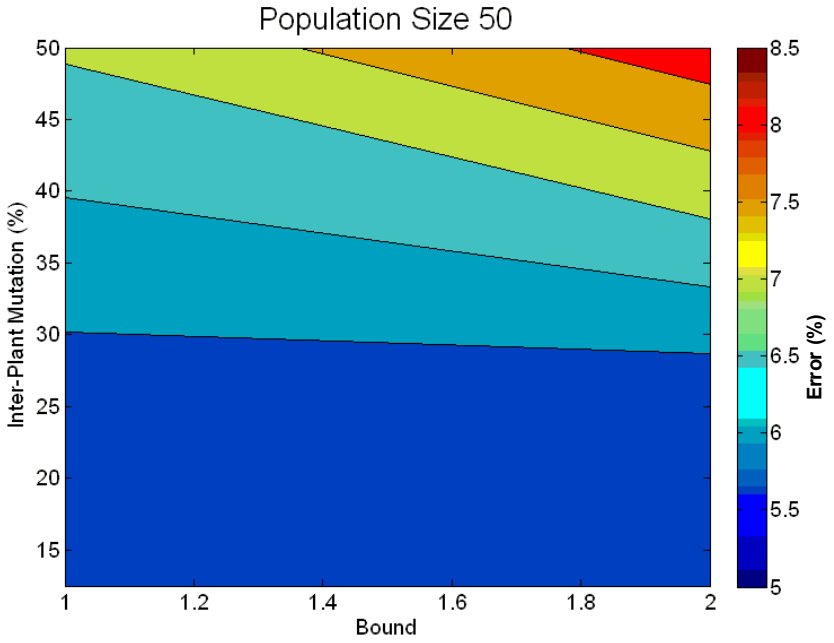

Figure 5-10 Genetic algorithm: Population Size 50

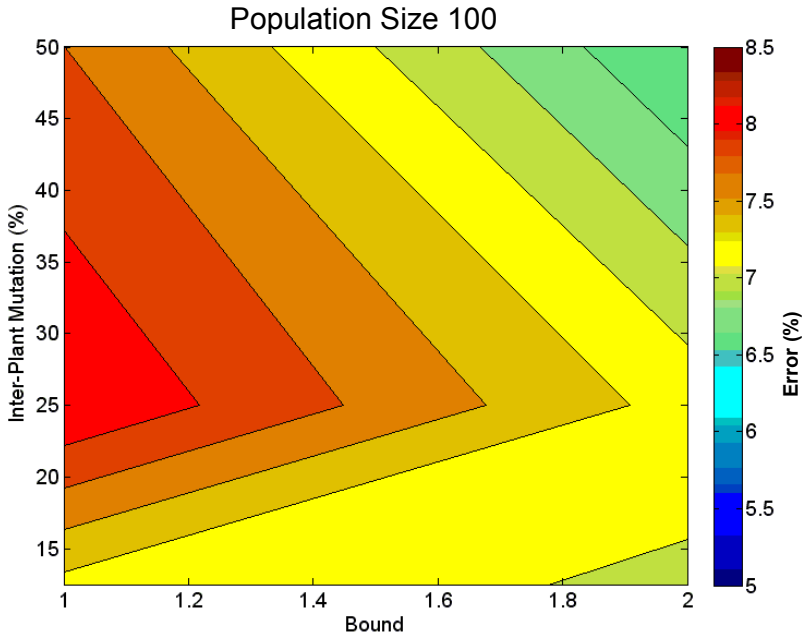

Figure 5-11 Genetic algorithm: Population Size 100

The first observation that can be done is that the results with a population of 100 are much worse than the ones with 25 and 50. With this in mind, the last figure will be disregarded and focus will be placed on the other two.

With a population of 25, the best results are obtained at low bound/moderate mutation and also at high bound/high mutation pairs. On the other hand, a population of 50 shows bad results for high bound/high mutation while displaying good ones at low bound/moderate mutation.

As both of the figures present good results at low bound/moderate mutation, this will be the parameters adopted as default.

In light of what was said, the optimal parameters chosen for the Genetic Algorithm are the following:

- Population Size: 25 (50% Number of collection sites)
- Inter-Plant Mutation: 25%
- Intra-Plant Mutation: 20%
- Bound: 1
- Elite Percentage: 4%

5.1.3. Ant Colony Optimization

For the Ant Colony Optimization algorithm described in 4.2.3, four parameters that might affect the solution quality (excluding the stopping criteria) will be discussed. There are another 3 parameters that will remain constant throughout testing. All will be described below:

- ***Number of colonies per iteration***

  This parameter corresponds to the number of colonies that set out to solve the problem at every iteration. As only the best colonies of each iteration contribute to dropping pheromone, a higher number of colonies will result in a more thorough exploration of the neighborhood, as more attempts are made before moving on to a different pheromone matrix. On the other hand, a higher computational effort is required at higher values leading to slower convergence. Values of 25, 50, 75 and 100 were tested.

- ***Heuristic Coefficient***

  The heuristic coefficient controls the weight given to the heuristic information matrix as opposed to the pheromone matrix. It sets the balance between previous knowledge and experience. Values of 0,6, 0,8 and 1 will be tested, for a fixed value of 1 for the pheromone coefficient.

- ***Evaporation Coefficient***

  Controls how fast the pheromone dropped by colonies will disappear. If it is too high the information supplied by the pheromones might not last long enough to be useful, but if it is too low the pheromone matrix might start to saturate and poor information from initial iterations might contribute to weaker solutions. Values of 0,1 and 0,2 will be tested.

- ***Bound***

  As in the Genetic Algorithm, the bound parameter is used only in the pre-processing phase of the algorithm, and determines how close a given site must be to a plant in order for the possibility of associating them to be considered. As stated in 4.2.3.1 though, this parameter is probably a lot more important to this algorithm when compared to the previous one. Values of 0,5 and 1 will be tested[10].

[10] Smaller values were chosen relative to the Genetic Algorithm due to the fact a large bound in Ant Colony Optimization could lead to ants straying too far from their plant, leading to very high return costs.

A total of 48 tests will be done with the following testing conditions:

- Maximum number of iterations: 1.000.000.000
- Maximum number of iterations without improvement: 1.000.000.000
- Maximum time spent: 1800s
- Pheromone Coefficient: 1
- Number of Pheromone Dropping Colonies: 1
- Pheromone to Drop: 0.1

Again, 1800s corresponds to the 30 minutes of computational time used before. The pheromone coefficient was fixed at 1 since this parameter is closely related to the heuristic coefficient, and changing both seemed unnecessary. The number of pheromone dropping colonies and pheromone to drop were fixed in order to reduce the number of parameters and because these values proved stable in previous testing.

Due to the difficulty in representing 4 varying parameters at the same time, the bound parameter was chosen to be examined first. Table 5-4 summarizes the average performance of tests under bound conditions.

Population Size	Bound	Average Error (%)
25	**0,5**	**9,86**
	1	12,29
50	**0,5**	**9,50**
	1	12,16
75	**0.5**	**9,22**
	1	12,99
100	**0.5**	**10,37**
	1	12,34

Table 5-4 Performance at different bound settings

As can be seen, the tests with bound set to 0.5 consistently outperform the ones set to 1. Also, a direct comparison between tests where the only parameter changed is this one shows that the 0.5 results are better in 21 out of 24 comparisons.

With that said, the bound parameter will be fixed at 0.5 and the remaining parameters will be plotted in the Figures 5-12 to 5-15.

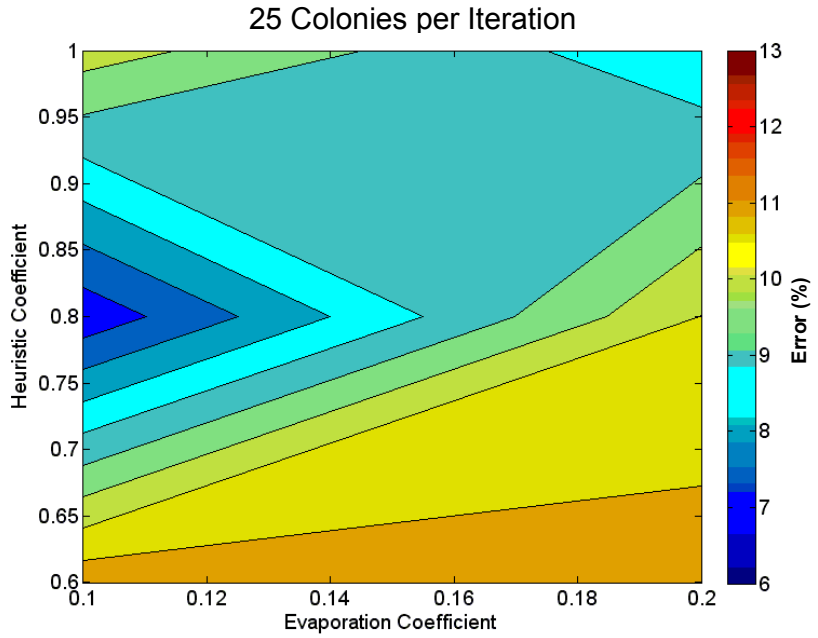

Figure 5-12 Ant Colony Optimization: 25 Colonies per Iteration

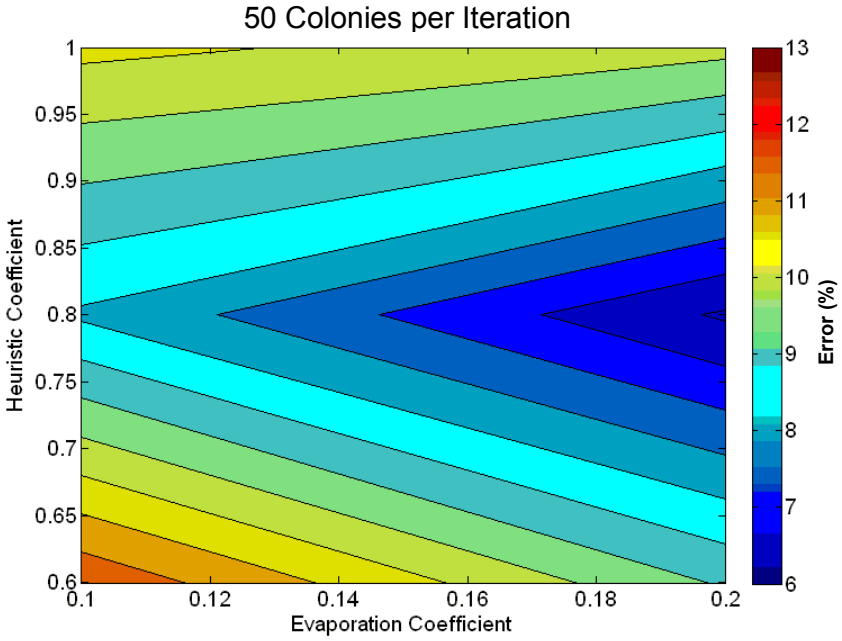

Figure 5-13 Ant Colony Optimization: 50 Colonies per Iteration

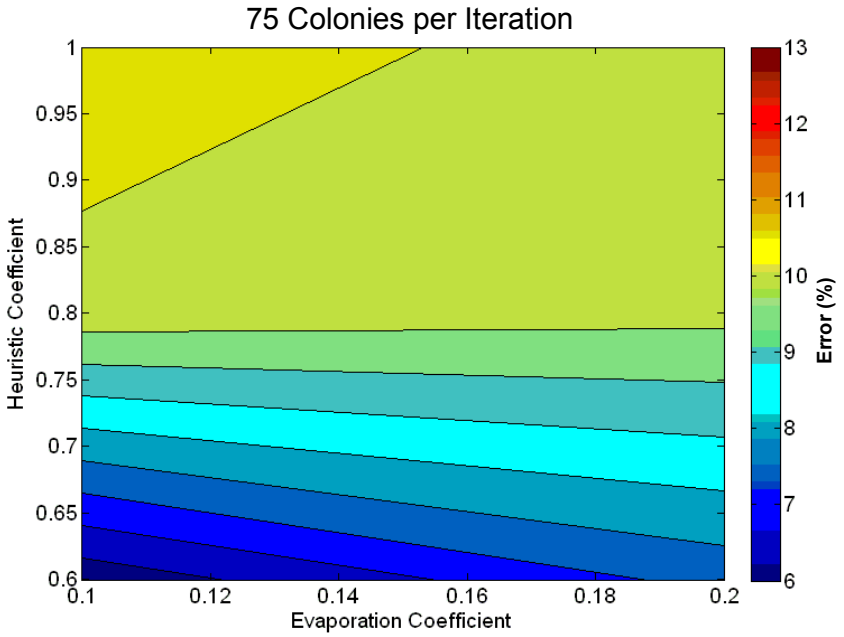

Figure 5-14 Ant Colony Optimization: 75 Colonies per Iteration

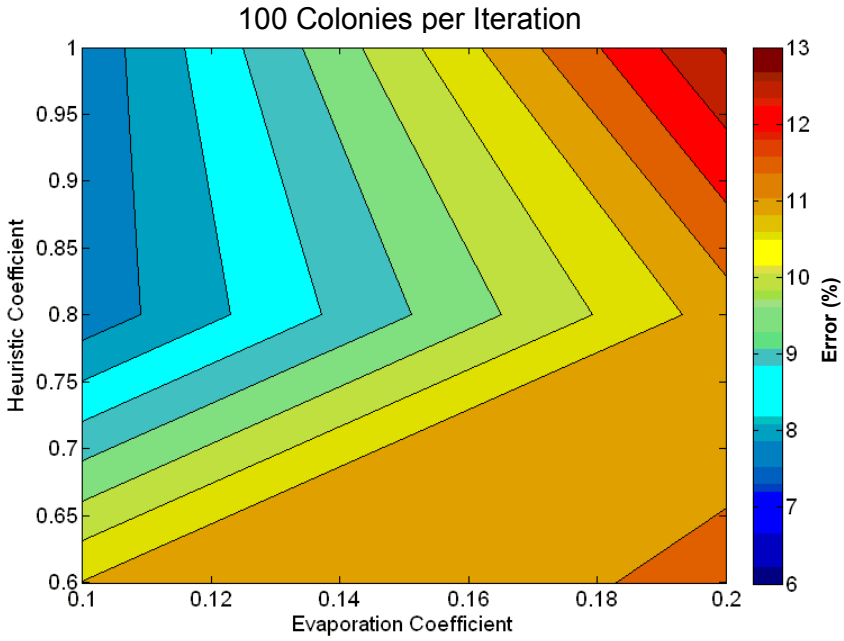

Figure 5-15 Ant Colony Optimization: 100 Colonies per Iteration

Focusing on the heuristic coefficient first, it can be seen that all but one figure shows its best results clustered around a value of 0,8 for this parameter.

As for the evaporation coefficient, it seems to influence the optimization significantly, but in different ways according to the number of colonies per iteration. At 25 and 100 colonies the results are much better at low levels of evaporation, while at 50 colonies better results are obtained at high evaporation but are still acceptable at low levels.

Lastly, the number of colonies that provide better results are 25 and 75, but the results obtained at 75 are inconsistent with the remaining figures in regards to heuristic and evaporation coefficients.

In light of what was said, the optimal parameters chosen for Ant Colony Optimization are the following:

- Number of Colonies per Iteration: 25 (50% Number of collection sites, as in Genetic Algorithm)
- Heuristic Coefficient: 0,8
- Evaporation Coefficient: 0,1
- Bound: 0,5
- Pheromone Coefficient: 1
- Number of Pheromone Dropping Colonies: 1
- Pheromone to Drop: 0,1

5.2. Benchmark Testing

The algorithms developed in Section 4.2 will now be tested using a few representative benchmark MDVRP instances from (Díaz 2007). Table 5-5 provides an overview of the instances that will be used.

Instance	Number of Sites	Number of Plants	Vehicle Capacity	Maximum Route Length	Best Known Solution
p01	50	4	80	-	576,87
p04	100	2	100	-	1001,59
p07	100	4	100	-	885,80
p09	249	3	500	310	3900,22
p13	80	2	60	200	1318,95

Table 5-5 Benchmark instances overview

The optimal parameters derived in 5.1 will be used in all cases. A series of 10 tests will be done for each algorithm/instance pair and statistical data will be collected. The stopping criteria will be 30 minutes of computational time for the Genetic Algorithm and Ant Colony Optimization and 29 minutes of fast improvement followed by a maximum of 100 iterations per route pair or 50 iterations without improvement per route pair of intensification for Tabu Search.

Table 5-6 summarizes the results obtained in the form of average error found (μ) in the 10 tests, standard deviation (σ), result of best test (Best) and average number of routes used[11] (μ_{Routes}).

[11] This number is before the post processing of section 4.3 as this is the usual value displayed in benchmark solutions

Instance	Tabu Search				Genetic Algorithm				Ant Colony Optimization			
	μ	σ	Best	μ_{Routes}	μ	σ	Best	μ_{Routes}	μ	σ	Best	μ_{Routes}
p01	2,96	1,24	0,79	12	14,29	1,63	11,16	11,8	9,66	1,37	7,56	**11,5**
p04	3,43	0,46	2,55	17	61,72	14,10	25,13	16	64,23	3,20	58,34	**15,4**
p07	1,29	0,22	1,05	17	41,81	12,21	9,19	17	47,26	2,17	42,01	**16,8**
p09	3,12	0,37	2,57	**26**	97,70	40,25	21,36	32,4	152,32	3,95	146,99	33,7
p13	1,37	0,84	0,0003	**8**	81,32	28,82	3,54	14,7	34,04	3,52	26,59	10

Table 5-6 Performance parameters for all algorithms on five benchmark problems

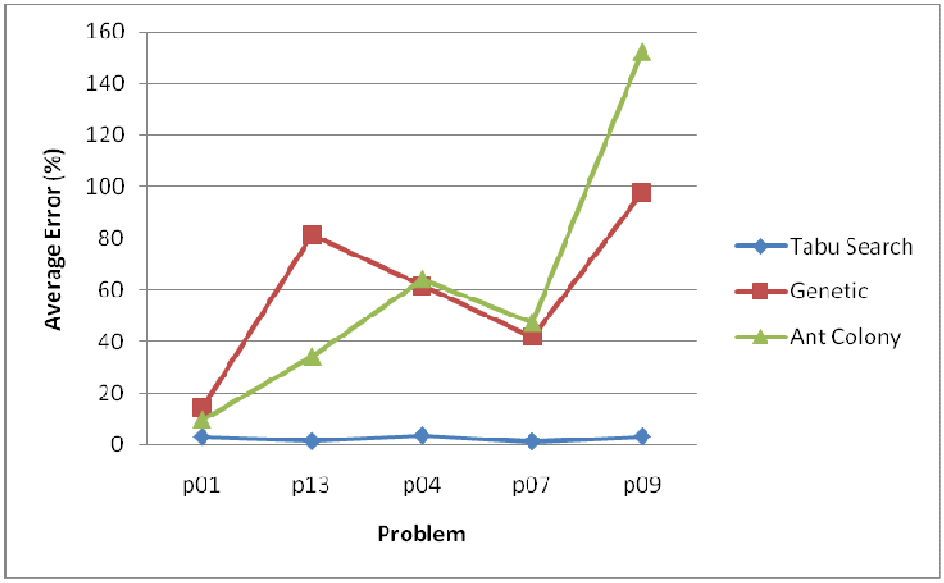

Figure 5-16 Performance of algorithms on all problems

As can be seen from both the table and the graphical analysis of the error in Figure 5-16, Tabu Search clearly dominates on all problem types. Both Genetic and Ant Colony algorithms provide acceptable results on the instance used for parameter tuning, p01, but their performance worsens considerably once the problem size starts to become larger. It was observed that, on larger problems, both algorithms do not manage to make enough iterations in the allocated time.

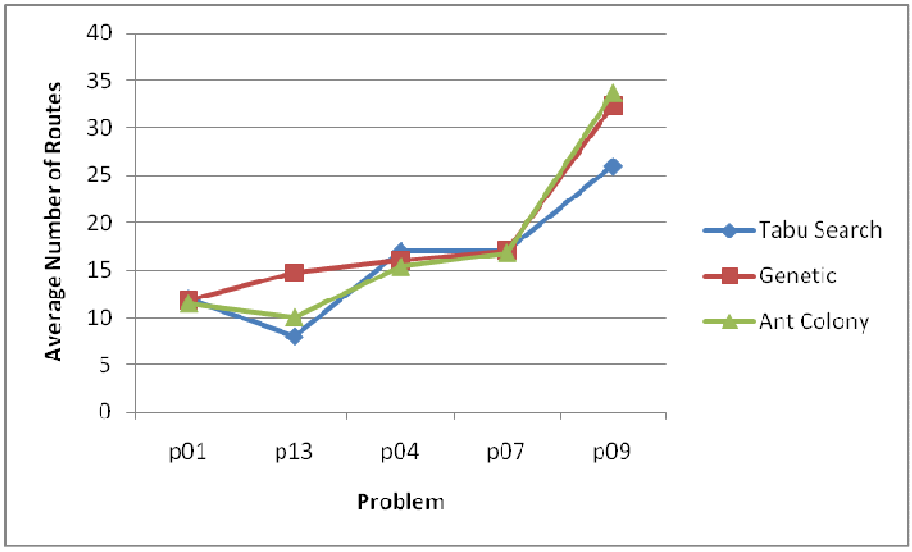

Figure 5-17 Average number of routes used per algorithm/problem pair

Figure 5-17 shows the evolution of the average number of routes (before post processing) used by each algorithm on each problem. All algorithms follow a similar pattern, though on some problems Ant Colony Optimization requires fewer routes on average than Tabu Search.

5.3. Application Testing

In this section the process of creating the Azores instance will be described first, followed by the results obtained after applying the algorithms to it.

5.3.1. Building the Instance

Due to the lack of information available on the distribution of biomass sources across the Azores islands, as well as the location of current and future biomass plants, a number of assumptions had to be made in order to make the instance creating feasible.

Firstly, the scope of the problem will be narrowed to the island of S. Miguel, due to the fact that the majority of the population (approximately 56% (Serviço Regional de Estatística dos Açores 2007)), and consequently waste production of the archipelago, is concentrated here. Additionally, only urban solid waste and forest residues will be considered as being useful to energy production.

To estimate the amount of solid urban waste across the island the projections for 2009 for total island population and total urban waste produced (Brito, et al. 2007) as well as the percentage population living in each municipality in 2007 (Serviço Regional de Estatística dos Açores 2007) were used. With this information the Table 5-7 was built:

Municipality	Population (2007)	Estimated Population (2009)	Estimated Waste Production (2009)	
			Per Year	Per Day
Lagoa	11,53%	15723	8994 ton	24,64 ton/day
Nordeste	3,97%	5414	3097 ton	8,49 ton/day
Ponta Delgada	48,20%	65730	37599 ton	103,01 ton/day
Povoação	5,10%	6955	3978 ton	10,90 ton/day
Ribeira Grande	22,84%	31147	17817 ton	48,81 ton/day
Vila Franca do Campo	8,36%	11401	6521 ton	17,87 ton/day

Table 5-7 Estimated population and urban waste production for S. Miguel

As for forest residues and again due to lack of data, the estimates presented in Table 5-8 were created using the locations of the main natural reserves on the island (Governo dos Açores 2009) as well as the total forest residues produced on the island in 2007.

Municipality	Natural Reserves	Total Area	Percentage Area	Estimated Forest Residues	
				Per Year	Per Day
Nordeste	Viveiro do Nordeste	1 ha	10,64%	3706 ton	10,15 ton/day
	Cancela do Cinzeiro	10 ha			
	Fajã do Rodrigo	1,5 ha			
Povoação	Viveiro das Furnas	3 ha	15,32%	5336 ton	14,62 ton/day
	Água Retorra	15 ha			
Lagoa	Chã da Macela	28 ha	23,83%	8301 ton	22,74 ton/day
Ponta Delgada	Pinhal da Paz	49 ha	41,70%	14526 ton	39,80 ton/day

Vila Franca do Campo	Cerrado dos Bezerros	10 ha	8,51%	2964 ton	8,12 ton/day

Table 5-8 Estimated forest residues for S. Miguel

Following this, the collection sites that correspond to each of municipalities will be placed, as there is no information on current or future planned collection sites. Sites will be assigned for each municipality for urban waste and forest residues if it also produces them. Most of these sites will be further split into more if their supply is large enough. The location of sites will be chosen respectively on the outskirts of urban areas or outside of them. As for plants, as no information exists at the moment, 3 locations will be chosen, one in the east, one in the west and one in the south of the island. Table 5-9 shows the locations and supplies of each site and plant and Figure 5-18 illustrates this graphically.

Municipality	Site Type	Site #	Latitude	Longitude	Daily Supply
Lagoa	Urban 1	1	37,750860º	-25,564358º	12,00 ton/day
	Urban 2	2	37,723177º	-25,516090º	12,26 ton/day
	Forest 1	3	37,770674º	-25,553545º	11,37 ton/day
	Forest 2	4	37,779241º	-25,597244º	11,37 ton/day
Nordeste	Urban	5	37,832583º	-25,151072º	8,49 ton/day
	Forest	6	37,845222º	-25,205272º	10,15 ton/day
Ponta Delgada	Urban 1	7	37,755609º	-25,676059º	25,75 ton/day
	Urban 2	8	37,885912º	-25,820059º	30,75 ton/day
	Urban 3	9	37,884551º	-25,731187º	3,50 ton/day
	Urban 4	10	37,831817º	-25,683493º	25,75 ton/day
	Urban 5	11	37,762268º	-25,628634º	22,25 ton/day
	Forest 1	12	37,785982º	-25,640598º	20,00 ton/day
	Forest 2	13	37,820745º	-25,702086º	9,4 ton/day
	Forest 3	14	37,815040º	-25,772195º	10,4 ton/day
Povoação	Urban	15	37,749289º	-25,245483º	10,90 ton/day
	Forest	16	37,755746º	-25,241752º	14,62 ton/day
Vila Franca do Campo	Urban 1	17	37,725072º	-25,440883º	12,44 ton/day
	Urban 2	18	37,719539º	-25,460319º	5,43 ton/day
	Forest	19	37,735366º	-25,436059º	8,12 ton/day
Ribeira Grande	Urban 1	20	37,813876º	-25,519549º	27,27 ton/day
	Urban 2	21	37,812337º	-25,574379º	7,56 ton/day
	Urban 3	22	37,817024º	-25,414346º	13,98 ton/day
-	West Plant	23	37,798255º	-25,696112º	-
-	East Plant	24	37,847936º	-25,256306º	-
-	South Plant	25	37,727821º	-25,471267º	-

Table 5-9 Location of S.Miguel's hypothetical sites and plants

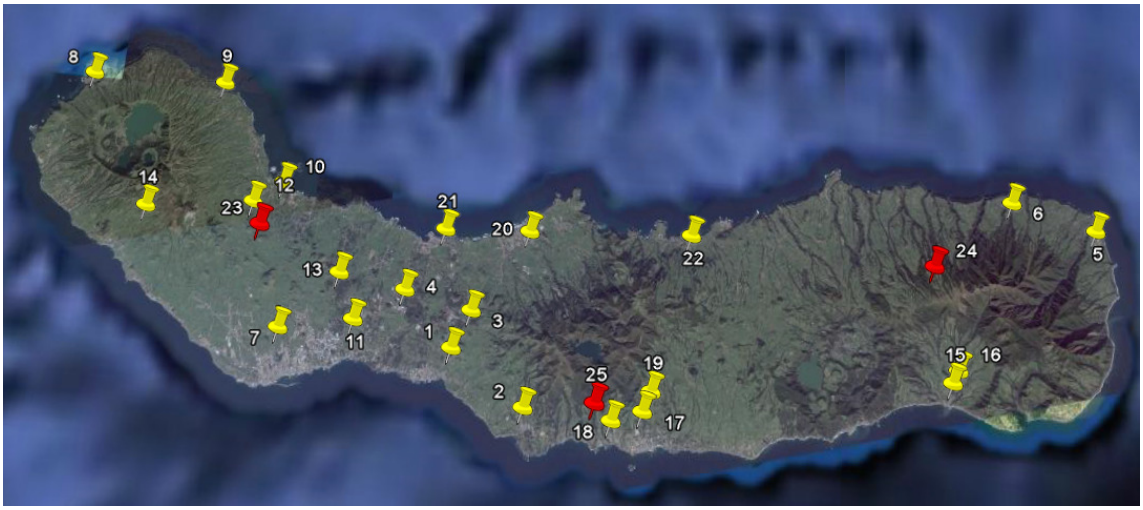

Figure 5-18 Location of S.Miguel's hypothetical sites (yellow) and plants (red)

As the application accepts coordinate data and not decimal degrees, the distances between locations were calculated using the equation (10)[12] followed by application of a 50% road non-linearity coefficient. The full distance matrix can be viewed in Azores_instance.xlsx, the file used to import data to the application

$$distance = r * \arccos\left[\sin(lat1) * \sin(lat2) + \cos(lat1) * \cos(lat2) * \cos(lon2 - lon1)\right] \quad (10)$$

where lat1 and lat2 correspond to the latitudes of the two locations and lon1 and lon2 to the longitudes. R is a constant that is equal to 6378,7, the earth's radius in kilometers.

As for the vehicles, a standard tipper truck was selected which can usually carry around 26000 kg, or 26 tons. Average speed will be set to 40 km/h and maximum length of route to 7 hours.

5.3.2. Results

A similar procedure to the one used in Section 5.2 will be used. A series of 10 tests will be done for each algorithm and statistical data will be collected. The stopping criteria will be 30 minutes of computational time for the Genetic Algorithm and Ant Colony Optimization and 29 minutes of fast improvement followed by a maximum of 100 iterations per route pair or 50 iterations without improvement per route pair of intensification for Tabu Search.

Table 5-10 summarizes the results obtained in the form of average time taken in minutes for the 10 tests, standard deviation, result of best test and average number of routes used.

Algorithm	μ (min)	σ (min)	Best (min)	μ_{Routes}
Tabu Search	587,445	0	587,445	15
Genetic Algorithm	587,671	0,714	587,445	15
Ant Colony Optimization	588,883	1,144	587,445	14,2

Table 5-10 Performance parameters for all algorithms on local problem

Analyzing the results, we can conclude that all the algorithms have shown good behavior on this problem. The small size of the instance, coupled with the fact that it was created according to the real

[12] The equation corresponds to the Great Circle Distance Formula from (Meridian World Data 2009). Degrees have to be converted to radians first.

data could account for this. As was observed before in the benchmark instances, Tabu Search produces the best and most consistent results while Ant Colony Optimization provides the least number of routes. This can be, in certain cases, more important than small time savings.

It is interesting to note that the Clark Wright Savings Heuristic on its own produces a solution with a total cost 587,445 minutes which might be presumed to be optimal from the above results. The best solution found is presented in graphical form below

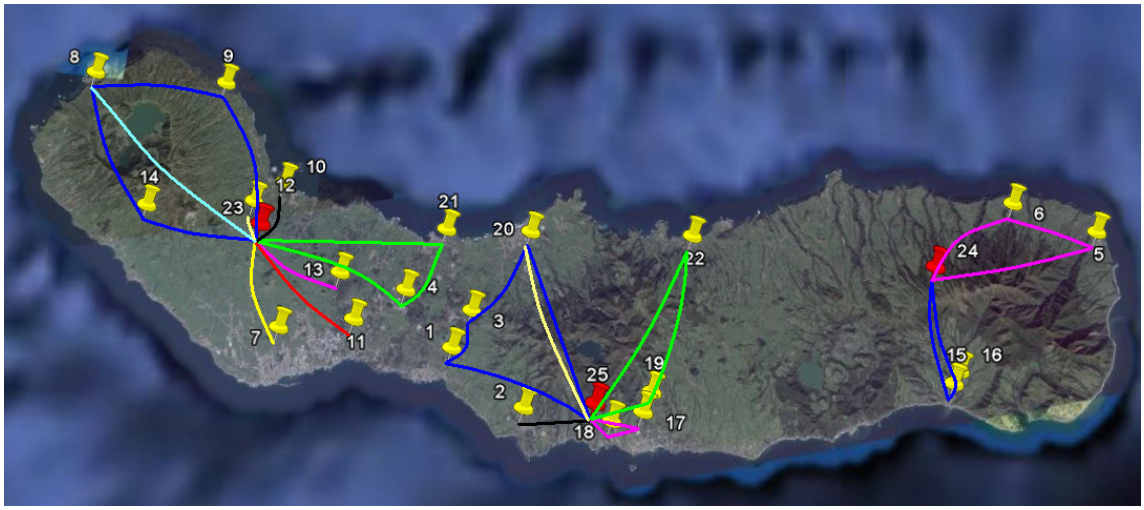

Figure 5-19 Graphical representation of best solution found

6. Conclusions and Future Work

The work describe above culminated in the development of the application for solving Multi-Depot Vehicle Routing Problems. It obtained relatively good results, with average errors as small as 1,3%, on the benchmarks tests used and easily reached a solution that is assumed optimal in the local problem.

The vast majority of good results were obtained using the Tabu Search heuristic which constitutes the approach that requires the least computational intensity. The computational intensity present in both the Genetic Algorithm (with the use of the route scheduler many times per iteration) and Ant Colony algorithm (with the constant evaluation of the attractiveness matrix) proved to be a large burden when these were applied to larger problems such as p09.

Still, the Ant Colony algorithm was able to find solutions that used fewer routes than Tabu Search which in itself can be considered a good result for something completely new that tried to respond to the lack of work in this field. On the local problem for example, it manages to find a solution with one fewer route at a negligible time cost. This, along with the fact that one of the weaknesses of the Tabu approach is its inability to alter the total number of routes present in its initial solution may present an opportunity for the development of a hybrid system between the two.

Even though the application developed is fully functional and could be applied to the original goal, optimizing the transport of biomass from collection sites to power plants, there are a number of improvements that could be made.

The introduction of additional sets of constraints on the problem should be addressed. Possibly the most important would be limiting the number of vehicles present at each plant that could be used for collection. Other constraints could be changed or added such as:

- Setting maximum and minimum amounts of mass required by each plant
- Allowing the use of vehicles with different characteristics (capacity, maximum travel time)
- Setting time windows for collection and delivery
- Setting penalties for failing to pick up or deliver a certain amount of mass

As for the optimization itself, and as mentioned above, an interesting approach would be to use the Ant Colony algorithm to initialize the Tabu Search approach, so that the low number of routes found by the ants could be further exploited.

7. References

Barbour, Ian, Harvey Brooks, Sanford Lakoff, and John Opie. *Energy and American Values.* California: Greenwood, 1982.

Basalla, George. "Energy and Civilization." *Science, Technology and the Human Prospect*, 1980: 39-52.

Battarra, M., R. Baldacci, and D. Vigo. *Clarke and Wright Algorithm.* Presentation. 2007.

Bella, John E., and Patrick R. McMullen. "Ant colony optimization techniques for the vehicle routing problem." Advance Engineering Informatics 18 (July 2004): 41-48.

Bridger, Mark. *Tabu Search.* Boston, November 2007.

Brito, António Guerreiro, et al. *Plano Estratégico de Gestão de Resíduos dos Açores.* Technical Report, Açores: Governo dos Açores, 2007.

Busetti, Frank. "Genetic Algorithms Overview." *Heuristics and Artificial Iintelligence in Finance and Investment.* November 9, 2000. www.geocities.com/francorbusetti/gaweb.pdf (accessed Setember 3, 2009).

Clark, G., and J. W. Wright. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points." *Operations Research Volume 12*, 1964: 568-581.

Climate Group. "In the black: The growth of the low carbon economy." ºClimate Group, London, United Kindom, 2006.

Cobb, Loren. "History & Future of World Energy." *The Quaker Economist.* March 27, 2007. http://www.quaker.org/clq/2007/TQE155-EN-WorldEnergy-1.html (accessed August 14, 2009).

Díaz, Bernabé Dorronsoro. *VRP Web.* March 2007. http://neo.lcc.uma.es/radi-aeb/WebVRP/ (accessed September 2009).

Dong, Wei Lin, and Cai Tian Xiang. "Ant Colony Optimization for VRP and Mail Delivery Problems." *IEEE International Conference on Industrial Informatics.* 2006. 1143-1148.

Dorigo, Marco. *Optimization, Learning, and Natural Algorithms.* Milan, 1992.

Garey, Michael R., and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* New York: W. H. Freeman & Co., 1990.

Glover, Fred, and Manuel Laguna. *Tabu search.* Boston: Kluwer Academic Publishers, 1997.

Governo dos Açores. *Direcção Regional dos Recursos Florestais.* September 24, 2009. http://www.azores.gov.pt/Portal/pt/entidades/sraf-drrf/?cName=sraf-drrf&lang=pt&area=ct (accessed September 24, 2009).

Hillier, Frederick S., and Gerald J. Lieberman. *Introduction to Operations Research.* McGraw Hill, 2001.

Holland, John H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT Press, 1975.

INETI. "AVALIAÇÃO DO POTENCIAL ENERGÉTICO." AREAM/INETI, Madeira, 2005.

Kanellos, Michael. "Can renewable energies make a dent in fossil fuels?" *CNET News.* April 25, 2008. http://news.cnet.com/8301-11128_3-9928068-54.html (accessed August 26, 2009).

Laporte, Gilbert. "The Vehicle Routing Problem: An overview of exact and approximate algorithms." *European Journal of Operations Research* 59 (1992): 345-358.

"Maps of the World." *Azerb.* http://azerb.com/maps.html (accessed August 2009, 27).

Marczyk, Adam. *Genetic Algorithms and Evolutionary Computation.* April 23, 2004. http://www.talkorigins.org/faqs/genalg/genalg.html#history (accessed August 31, 2009).

Medaglia, Andrés L., and Eliécer Gutiérrez. *The Capacitated Vehicle Routing Problem.* Presentation. Universidad de los Andes, January 23, 2005.

Meridian World Data. *Distance Calculation.* October 4, 2009. http://www.meridianworlddata.com/Distance-Calculation.asp (accessed October 4, 2009).

Merriam-Webster. *Webster's Dictionary.* 2009.

Obitko, Marek. *Introduction to Genetic Algorithms.* September 1998. http://www.obitko.com/tutorials/genetic-algorithms/index.php (accessed September 9, 2009).

Ombuki-Berman, Beatrice, and Franklin T. Hanshar. "Using Genetic Algorithms for Multi-depot Vehicle Routing." Chap. 4 in *Bio-inspired Algorithms for the Vehicle Routing Problem*, by Francisco Baptista Pereira and Jorge Tavares, edited by Janusz Kacprzyk, 77-99. Warsaw: Polish Academy of Science, 2008.

Pinedo, Michael L. *Planning and Scheduling in Manufacturing and Services.* Springer, 2005.

Pote, Michael. "Genetic Algorithms." *Nitrogen.* June 3, 2006. http://www.nitrogen.za.org/viewtutorial.asp?id=4 (accessed August 31, 2009).

Rechenberg, Ingo. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Berlin: Fromman-Holzboog, 1973.

Renaud, Jacques, Fayez F. Boctor, and Gilbert Laporte. "A fast composite heuristic for the symmetric traveling salesman problem." *INFORMS* 8 (January 1996-A): 134-143.

Renaud, Jacques, Fayez F. Boctor, and Gilbert Laporte. "An improved petal heuristic for the vehicle routing problem." *Journal of the Operational Research Society* 47 (June 1996-B): 329-336.

Renaud, Jacques, Gilbert Laporte, and Fayez F. Boctor. "A tabu search heuristic for the multi-depot vehicle routing problem." Computer Ops Research 23 (April 1995): 229-235.

Rentizelas, A. A., I. P. Tatsiopoulos, and A. Tolis. "An optimization model for multi-biomass tri-generation." Biomass and Energy 33 (May 2008): 223-233.

Serviço Regional de Estatística dos Açores. "Statistical Yearbook of the Azores Region." Açores, 2007.

Shekhawat, Anirudh, Pratik Poddar, and Dinesh Boswal. "Ant colony Optimization Algorithms: Introduction and Beyond." *Department of Computer Science and Engineering: Indian Institute of Tecnology Bombay.* March 27, 2009. http://www.cse.iitb.ac.in/~pratik/projectreports/aco.pdf (accessed August 31, 2009).

Silva, Carlos Augusto Santos. "Distributed Supply Chain Management using Ant Colony Optimization." PhD Dissertation, IST, Technical University of Lisbon, 2005.

Stidsen, Thomas. "TABU search and Iterated Local Search." *DTU Informatics - Technical University of Denmark.* February 17, 2008. http://www2.imm.dtu.dk/courses/02719/tabu/4tabu2.pdf (accessed August 31, 2009).

Stutzle, Thomas. *Ant Colony Optimization: An Introduction.* Gottingen, April 20, 2005.

Toth, Paolo, and Daniele Vigo. *The Vehicle Routing Problem.* Philadelphia: S.I.A.M, 2002.

Williams, James C. *The History of Energy.* April 25, 2006. http://www.fi.edu/learn/case-files/energy.html (accessed September 2009).

World Nuclear Association. "Renewable Energy and Electicity." *World Nuclear Association.* June 2009. http://www.world-nuclear.org/info/inf10.html (accessed August 26, 2009).

Xu, Jiefeng, and P. James Kelly. "A Network Flow-Based Tabu Search Heuristic For The Vehicle Routing Problem." *Transportation Science* 30 (February 1996): 379-393.

A. DailyPlan's User Manual

Welcome to DailyPlan's User Manual. In this manual the user will be given a step-by-step explanation on how to best use the application to solve a Multi-Depot Vehicle Routing Problem. The manual will be broken down into the following sections for easier reading:

A1. Creating the problem instance

A2. Importing the instance

A3. Solving

A4. Exporting results

A1. Creating the Problem Instance

To create the problem instance, the user must be able to manipulate .xlsx files by use of Microsoft Excel or a similar application. Open the file named InstanceTemplate.xlsx. In it, the template for building problem instances can be found.

The workbook contains three sheets, Coordinates, Distances and Times. Each of these constitutes a separate way of introducing problem data. The application assumes that times are more precise than distances and that both are more precise than coordinates and, in case multiple sources are available, will use the one deemed most precise.

If times are to be used, the corresponding sheet should be filled in as shown in the Figure A-1. The boxes with the number of collection sites and plants should be correctly filled in, followed by the names of first sites and then plants horizontally and vertically expanding from the "Names" cell. The interior should then be filled in with the time taken in minutes between all sites and plants (fractional values may be used according to .xls format). While only the top triangle of the matrix is relevant to the application, the bottom left can also be written for completeness. Below the last plant a supply row should be constructed, containing the supplies from all collection sites inserted.

Fill "Names" row and colunm with the names of collectionsites and plants in that order.
Fill the upper diagonal with time taken in trips between them in minutes.

Number of Collection Sites	5
Number of Plants	3

Names	CS X123	CS Y872	CS Z981	CS B93	CS R32	West Plan	East Plant	Northwest Plant
CS X123		87	12	56	54	34	31	7
CS Y872			32	54	65	78	45	5
CS Z981				23	43	54	76	6
CS B93					23	1	234	3
CS R32						32	323	5
West Plant							44	6
East Plant								13
Northwest Plant								
Supply	33	23	23	13	11			

Figure A-1 Example of filled in time sheet

The process is exactly the same for using distances, except for the sheet that is used.

If coordinates are to be used, the corresponding sheet should be filled in as shown in Figure A-2. There is no need to explicitly declare the number of collection sites or plants in this case. The names, coordinates and supplies (if applicable) should be filled for all sites and plants.

All measures in kilometers

Collection	*Name*	*Latitude*	*Longitude*	*Supply*	**Plants**	*Name*	*Latitude*	*Longitude*
	CS X123	22	33	33		North Plant	33	12
	CS Y872	11	31	23		South Plant	12	33
	CS Z981	21	55	23		West Plant	76	46
	CS B93	33	77	50				
	CS R32	21	20	12				
	CS P01	11	1	5				

Figure A-2 Example of filled in coordinate sheet

The workbook should now be saved with a different name so that the template is maintained for future work. That is it for creating the problem instance. The user can now proceed with importing it to the application.

A2. Importing the Instance

In order to import the instance, the application must first be started. Double-click DailyPlan.fig or open MATLAB and type DailyPlan in the workspace (don't forget to set your working directory to the folder where the application is located). The main menu will appear as shown in Figure A-3 and the only available button, *Import*, should be clicked.

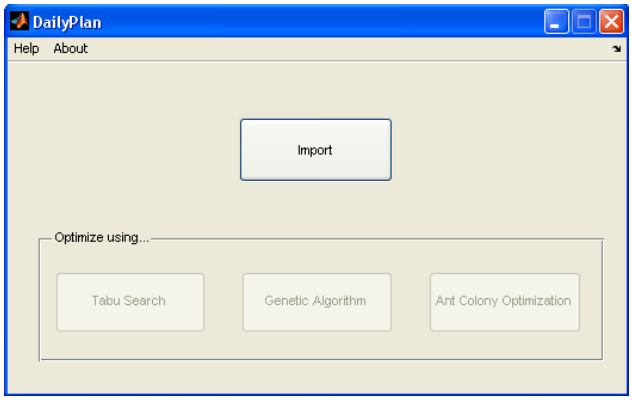

Figure A-3 DailyPlan: Main window

A new window will appear. Here, the user should click *Load* and select the previously created workbook (or any workbook in the template format). A progress bar as well as text under the button will indicate the status of the loading of data. Once loading as been completed the *Average Truck Speed* and *Road Non-Linearity Coefficient* might be available for filling in[13]. They will only become available if they are relevant to the data at hand as shown in Figure A-4. If time data was loaded none are needed, while both are required to convert coordinate data into time. If distance data is loaded, only the average truck speed will be required.

[13] Many benchmark MDVRP problems are supplied with data in coordinates and the cost function being distance travelled. In this case, the average truck speed should be set to 60 km/h and road non-linearity coefficient to 0%.

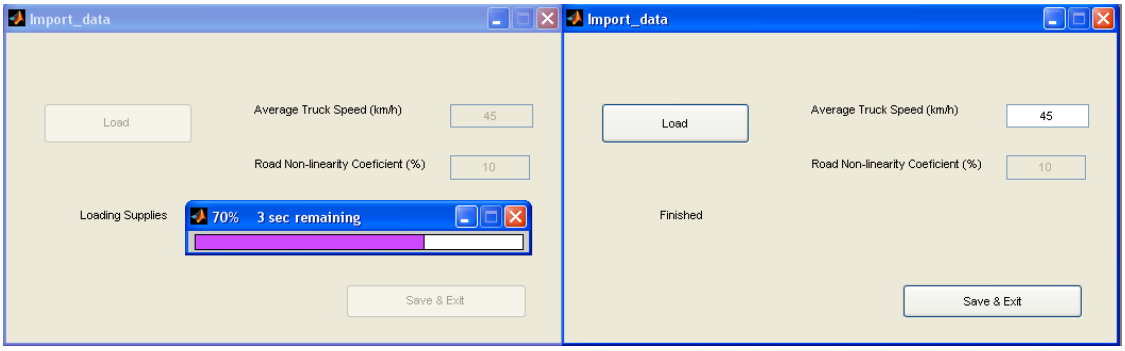

Figure A-4 On the left, importing data. On the right, distance data has been imported (only speed available)

Pressing *Save & Exit* will return the user to the main menu, while clicking the *X* in the top right corner will cancel the import. If for any reason *Save & Exit* is pushed before a data is successfully imported, the application will use the last successfully imported problem data. While in the main window, the user can choose to go back and import a different problem (in case of mistakes for example) or proceed to the next section.

A3. Solving

If data was imported successfully, the buttons under *Optimize using…* will become available for pushing. The user should push the one corresponding to the solving method they wish to use for this run. A new window will open revealing the parameters that should be set before optimization can begin. Figure A-5 shows an example of the Genetic Window.

The parameters are divided into 3 classes: Problem Data, Stopping Criteria and Advanced Settings. The first two are mostly equal for all 3 solving techniques while Advanced Settings are specific to each.

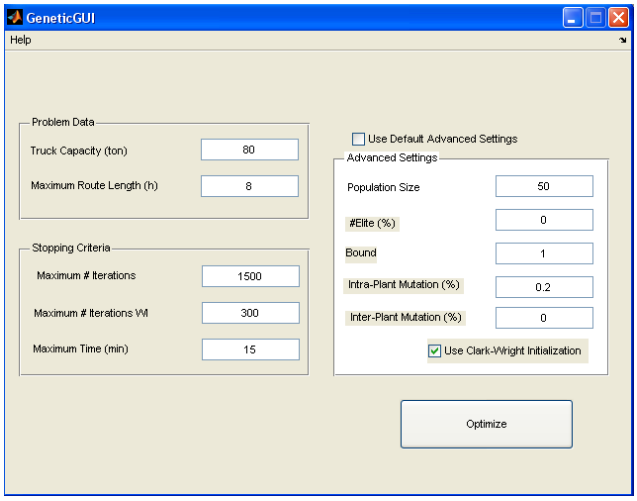

Figure A-5 Example of Genetic Algorithm user interface

A3.1. Problem Data and Stopping Criteria

In the problem data section, the truck capacity and maximum route length should be introduced.

- Truck Capacity – The maximum amount of weight that a truck is capable of transporting in tons.

- Maximum Route Length – The maximum amount of time in hours that a truck can travel.

In the stopping criteria section, the maximum number of iterations, maximum number of iterations without improvement and maximum time taken should be introduced.

- Maximum number of iterations – The maximum number of iterations that the algorithm is allowed to use.
- Maximum number of iterations without improvement – The number of iterations without improvement of the best solution at which the algorithm breaks.
- Maximum time – The maximum time in minutes that can be spent in on optimization.

The Tabu Search algorithm, due to having 2 distinct phases, has a total of five stopping criteria instead of 3.

Whenever one of these parameters is left at 0 the application assumes that it should not be used and its value will default to infinity.

At least one item in the problem data section must be filled in (or the problem would be unconstrained) and at least one item (two for Tabu Search, one for each phase) must be filled in the stopping criteria section (or the algorithm would run forever). Allowable and recommended ranges for all these parameters can be found in the corresponding help page.

A3.2. Advanced Settings

The Advanced Settings section is locked by default. To unlock it, the *Use Default Advanced Settings* box should be unchecked. Advance settings are specific to each problem solving technique.

- **Tabu Search**

  Tabu Search contains the following advanced parameters:

  - Local optimization maximum number of iterations – The maximum number of iterations allowed for local optimization.
  - Tabu size – The size of the tabu list.
  - Probability – The probability of choosing the closest plant in inter-plant exchanges.
  - Number of Sub-Cycles – The number of sub-cycles in each iteration.

- **Genetic Algorithm**

  The Genetic Algorithm contains the following advanced parameters:

  - Population size – The number of chromosomes in each generation.
  - Number of elite – The number of elite individuals cloned to the next generation in percentage of total number of chromosomes.
  - Bound – Controls how close a collection site must be to a plant for them to be possibly assigned to each other.
  - Intra-plant mutation – The percentage chance of intra-plant mutation happening at each generation.
  - Inter-plant mutation – The percentage chance of inter-plant mutation happening at each generation.

- **Ant Colony Optimization**

  The Ant Colony Optimization algorithm contains the following advanced parameters:

  - Pheromone Coefficient – The relative importance of pheromone information.
  - Heuristic Coefficient – The relative importance of heuristic information.
  - Evaporation Coefficient – The percentage amount of pheromone that evaporates at each iteration.
  - Number of Pheromone Dropping Colonies – The number of colonies that leave pheromone in their trail at each iteration.
  - Colonies/Attempts per Iteration – The number of colonies sent out at each iteration.
  - Bound – Controls how close a collection site must be to a plant for them to be possibly assigned to each other.

Whenever one of these parameters is left at 0 or if the *Use Default Advanced Settings* box is left checked the application uses the default value.

Allowable and recommended ranges for all these parameters can be found in the corresponding help page.

Once all required parameters have been set the optimization can begin as soon as the user presses the *Optimize* button.

A4. Exporting Results

After the optimization process has finished, the user will be prompted with a dialog box containing information on the total cost of the best solution found and requesting permission to proceed with the exporting of results (Figure A-6). In case the user wishes to export the data, a writable .xls or .xlsx file will need to be selected as well as a sheet name.

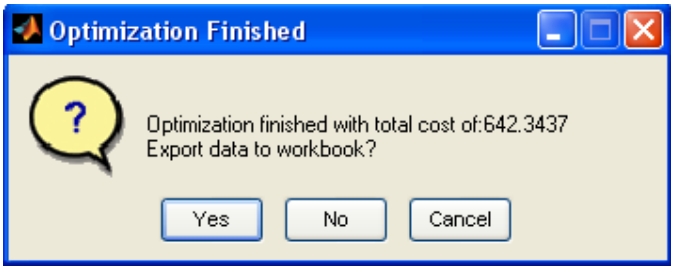

Figure A-6 Example of prompt after optimization has finished

The application will export the solution data to the selected worksheet as displayed in Figure A-7. For each route 3 rows will be displayed containing respectively the ordered list of collection sites or plants to visit, the required amount of material to pick up at each (a 0 means that a plant is being visited and all material should be unloaded) and the cost of traveling between the previous location and the current one. Below the last route, the total cost is also displayed for reference.

	A	B	C	D	E	F	G	H	I	J
1										
2	Location Name			S1	S4	S5	S3	S10	S1	S7
3	Amount to Pickup			0	21	28	23	7	0	3
4	Travel Cost			0	23,4094	6,403124	13,15295	19	20,51828	7,615
5										
6	Location Name			S2	S8	S3	S3	S7	S4	S6
7	Amount to Pickup			0	17	16	11	19	10	5
8	Travel Cost			0	15,81139	9,219544	13,0384	12,04159	14,21267	27,29
9										
10	Location Name			S3	S10	S3	S5	S5	S5	S3
11	Amount to Pickup			0	5	23	10	10	21	0
12	Travel Cost			0	9,055385	12,08305	7	6,708204	14,56022	10
13										
14	Location Name			S4	S8	S1	S6	S8	S1	S2
15	Amount to Pickup			0	14	11	7	23	7	8
16	Travel Cost			0	24,04163	6,324555	10,04988	7,211103	11,6619	7,071
17										
18	Total Cost			642,3437						

Figure A-7 Example of data exported to worksheet

B. Parameter Tuning Tables

Tables B-1 to B-4 contain the data used to plot the graphical analysis of Section 5.1.

In all tables, the error is calculated relative to the best know solution of the problem[14]. The percentage of useful iterations is obtained by finding the fraction of iterations of fast improvement that were required to obtain a solution value that is within 0,05% of the final solution found (a dash means that no improvement was done during fast improvement).

The first two tables are relative to the Tabu Search algorithm with Clark Wright initialization and improved petal initialization respectively. All solutions found use 13 and 12 routes respectively, as the algorithm does not change the number of total routes supplied by the initial solution.

Tabu Size	Probability	Sub-Cycles	# Fast Improvement Iterations	# Intensification Iterations	Useful Iterations (%)	Error (%)
25	0,3	5	101373	873	-	10,97
		20	27746	826	28,29	4,99
		35	15224	916	74,85	4,07
		50	10633	800	11,38	2,17
	0,6	5	102679	870	2,30	6,59
		20	26547	849	53,56	2,61
		35	14826	895	9,07	2,23
		50	10756	800	99,43	3,93
	0,9	5	91735	963	-	10,53
		20	29072	800	10,68	9,65
		35	15828	839	6,59	5,17
		50	10539	800	32,61	2,17
50	0,3	5	100575	859	-	10,74
		20	26341	820	17,52	3,33
		35	15120	826	9,91	4,37
		50	10475	836	75,90	2,17
	0,6	5	103629	837	90,55	6,86
		20	26370	839	8,28	3,42
		35	15683	820	19,36	2,17
		50	11040	872	31,60	2,17
	0,9	5	102814	911	0,00	9,86

[14] $Error = \frac{Solution - BestKnow}{BestKnow} * 100$

		20	27341	844	34,93	3,34
		35	15089	822	31,99	2,23
		50	10729	800	36,62	5,47
75	0,3	5	99648	856	-	10,90
		20	25414	817	9,07	5,03
		35	15487	802	5,48	5,73
		50	10499	802	66,02	2,17
	0,6	5	97061	856	-	12,11
		20	27544	820	10,60	3,65
		35	15130	823	2,07	3,33
		50	10583	800	16,31	4,33
	0,9	5	100292	859	0,00	9,97
		20	27062	864	8,79	2,23
		35	15929	800	10,42	6,29
		50	10141	800	56,58	4,33
100	0,3	5	100135	857	-	10,85
		20	25875	837	18,93	4,35
		35	14579	831	7,11	3,00
		50	10183	846	89,70	2,17
	0,6	5	95038	821	-	11,08
		20	28451	883	1,64	6,10
		35	14670	820	8,27	4,67
		50	9911	800	70,86	2,17
	0,9	5	94702	840	-	11,39
		20	25697	801	14,18	2,23
		35	15838	867	14,65	2,63
		50	10202	843	43,61	3,04

Table B-1 Tabu optimization results with Clark Wright initialization

Tabu Size	Probability	Sub-Cycles	# Fast Improvement Iterations	# Intensification Iterations	Useful Iterations (%)	Error (%)
25	0,3	5	90643	863	-	7,85
		20	23887	681	46,50	0,89
		35	13444	702	11,74	2,55
		50	9285	656	91,85	0,89
	0,6	5	86119	751	47,51	5,95
		20	24129	700	7,68	5,29
		35	13835	658	78,40	1,08
		50	9464	650	64,38	2,36
	0,9	5	86829	752	-	9,07
		20	22210	696	40,18	0,79

		35	12653	650	50,58	0,79
		50	9604	726	81,72	2,87
50	0,3	5	89282	732	-	7,74
		20	23302	650	59,27	1,04
		35	13197	650	95,02	1,02
		50	9374	650	94,55	1,33
	0,6	5	83591	671	-	8,55
		20	22427	652	39,85	2,53
		35	14005	650	16,89	3,85
		50	9169	650	46,23	0,79
	0,9	5	83233	768	-	8,55
		20	22348	650	86,58	4,29
		35	13800	650	39,29	0,79
		50	9854	650	41,35	1,02
75	0,3	5	88530	710	-	8,55
		20	22952	710	54,20	1,02
		35	13574	702	35,15	3,13
		50	9416	668	9,13	0,79
	0,6	5	90018	698	-	8,44
		20	23649	688	84,70	4,55
		35	13820	650	79,37	0,79
		50	9141	672	42,88	2,71
	0,9	5	84034	766	-	6,53
		20	24667	703	76,31	7,06
		35	13349	650	89,74	3,60
		50	8928	651	69,80	3,56
100	0,3	5	83276	692	-	9,26
		20	22736	676	13,97	0,79
		35	13625	673	93,86	4,06
		50	9221	659	8,29	0,79
	0,6	5	86740	740	-	8,44
		20	22706	658	39,65	2,10
		35	12960	650	97,58	0,79
		50	9256	659	50,25	1,63
	0,9	5	86972	675	-	8,55
		20	22968	650	37,49	1,27
		35	12954	666	99,59	1,30
		50	9153	650	75,10	0,79

Table B-2 Tabu optmization results with Improved Petal initialization

The next table shows the results obtained for the Genetic Algorithm. Four parameters will be tested and the number of vehicles used is now an important result as it might change from test to test.

The useful iterations percentage indicates the fraction of iterations required to reach a solution that was within 0,05% of the best found.

Population Size	Inter-Mutation (%)	Intra-Mutation (%)	Bound	Useful Iterations (%)	# Vehicles Used	Error (%)
25	12,5%	10%	1	47,51	12	8,01
			2	83,59	12	10,34
		20%	1	97,58	12	6,43
			2	96,39	12	6,45
	25%	10%	1	92,57	12	8,24
			2	69,21	12	6,25
		20%	1	71,43	12	5,61
			2	94,05	12	6,40
	50%	10%	1	89,92	12	9,45
			2	92,42	12	9,19
		20%	1	48,40	12	6,13
			2	95,36	12	5,24
50	12,5%	10%	1	80,18	12	6,39
			2	84,08	12	8,38
		20%	1	47,03	12	5,75
			2	86,30	12	5,61
	25%	10%	1	93,84	12	7,98
			2	80,43	12	8,18
		20%	1	85,56	12	5,72
			2	47,70	12	5,61
	50%	10%	1	89,07	12	10,80
			2	79,32	12	5,72
		20%	1	89,77	12	7,06
			2	83,86	12	8,27
100	12,5%	10%	1	91,71	12	10,83
			2	87,30	12	5,95
		20%	1	96,84	12	7,34
			2	82,01	12	7,16
	25%	10%	1	95,29	12	8,78
			2	87,25	12	8,78
		20%	1	95,51	12	8,19
			2	87,46	12	7,32
	50%	10%	1	98,71	12	9,27
			2	97,07	12	8,19
		20%	1	90,41	12	7,80
			2	92,67	12	6,60

Table B-3 Genetic Algorithm optimization results

The last table shows results of testing for the Ant Colony Optimization. Similarly to the Genetic Algorithm, the number of vehicles used is also shown as it might vary between tests. The useful iterations percentage indicates the fraction of iterations required to reach the best solution found.

Colonies per iteration	Heuristic Coefficient	Evaporation Coefficient	Bound	Useful Iterations (%)	# Vehicles Used	Error (%)
25	0,6	0,1	0,5	80,43	11	11,35
			1	98,01	12	12,97
		0,2	0,5	32,67	11	11,28
			1	62,82	12	16,51
	0,8	0,1	0,5	67,64	12	7,16
			1	65,82	11	11,47
		0,2	0,5	56,40	11	10,51
			1	40,29	12	12,53
	1	0,1	0,5	95,90	12	10,24
			1	53,26	12	9,08
		0,2	0,5	55,48	12	8,59
			1	73,91	11	11,16
50	0,6	0,1	0,5	97,68	11	11,90
			1	94,53	11	11,77
		0,2	0,5	96,12	11	9,44
			1	79,93	11	18,93
	0,8	0,1	0,5	43,86	12	8,42
			1	61,89	11	9,31
		0,2	0,5	83,81	11	6,43
			1	33,59	11	11,79
	1	0,1	0,5	73,90	11	10,63
			1	54,99	12	10,91
		0,2	0,5	57,85	12	10,16
			1	42,49	11	10,27
75	0,6	0,1	0,5	75,24	11	6,16
			1	89,49	11	10,74
		0,2	0,5	68,83	11	7,68
			1	97,64	11	12,05
	0,8	0,1	0,5	66,24	11	10,28
			1	98,44	11	7,59
		0,2	0,5	77,09	11	10,14
			1	39,53	11	20,56
	1	0,1	0,5	99,37	11	10,85
			1	92,94	11	12,58
		0,2	0,5	34,56	11	10,19
			1	49,03	12	14,39
100	0,6	0,1	0,5	97,93	12	11,02

			1	97,34	12	16,19
		0,2	0,5	81,93	12	11,60
			1	46,67	12	14,17
	0,8	0,1	0,5	81,85	11	7,68
			1	95,81	11	8,92
		0,2	0,5	46,35	11	11,24
			1	88,48	11	12,47
	1	0,1	0,5	91,80	12	7,65
			1	97,41	12	9,50
		0,2	0,5	44,40	11	13,05
			1	86,36	11	12,76

Table B-4 Ant Colony Optimization results