UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

DESIGN OF A SELECTIVE PARALLEL HEURISTIC ALGORITHM FOR THE VEHICLE ROUTING PROBLEM ON AN ADAPTIVE OBJECT MODEL

ALWYN JAKOBUS MOOLMAN

A thesis submitted in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY (INDUSTRIAL SYSTEMS)

In the

FACULTY OF ENGINEERING, BUILT ENVIRONMENT AND INFORMATION TECHNOLOGY

UNIVERSITY OF PRETORIA

APRIL 2010

ACKNOWLEDGEMENTS

To my family who supported me

To my mentor who inspired me

To my colleagues who endured me

To my twins who motivated me

To my wife who loves me

To God

ABSTRACT

Title: Design of a Selective Parallel Heuristic Algorithm for the Vehicle Routing Problem on an Adaptive Object Model.

Author: Alwyn Jakobus Moolman
Promoter: Professor VSS Yadavalli
Department: Industrial and Systems Engineering
University: University of Pretoria
Degree: Doctor of Philosophy (Industrial Systems)

The Vehicle Routing Problem has been around for more than 50 years and has been of major interest to the operations research community. The VRP pose a complex problem with major benefits for the industry. In every supply chain transportation occurs between customers and suppliers.

In this thesis, we analyze the use of a multiple pheromone trial in using Ant Systems to solve the VRP. The goal is to find a reasonable solution for data environments of derivatives of the basic VRP. An adaptive object model approach is followed to allow for additional constraints and customizable cost functions. A parallel method is used to improve speed and traversing the solution space. The Ant System is applied to the local search operations as well as the data objects. The Tabu Search method is used in the local search part of the solution.

The study succeeds in allowing for all of the key performance indicators, i.e. efficiency, effectiveness, alignment, agility and integration for an IT system, where the traditional research on a VRP algorithm only focuses on the first two.

Key words: Vehicle Routing Problem; Meta-heuristics, Hyper-heuristics, Memetic Algorithm, Ant System, Tabu Search; Multiple constraints; Multiple Time Windows; Supply Chain Management; Compatibility Matrix, Parallel

Table of Contents

Table of Figures

Table of Tables

Table of Equations

Table of Algorithms

Glossary

ABM	Agent-based Models
ACO	Ant Colony Optimisation
AOM	Adaptive Object Model
AMP.	Adaptive Memory Programming.
GA	Genetic algorithms
IRP	Inventory Routing Problem
MA	Memetic Algorithm
MAP	Meta-heuristic Agent Processes
MDVRP	Multi-Depot Vehicle Routing Problem
NP-hard	Non-polynomial hard
PFSIH	Push Forward Sequential Insertion Heuristic
SA	Simulate annealing
SIH	Sequential Insertion Heuristic
TS	Tabu search
TSP	Travelling Salesmen Problem
VFM	Vehicle Fleet Mix
VRP	Vehicle Routing Problem
VRPHE	Vehicle Routing Problem with Heterogeneous Fleet
VRPM	Vehicle Routing Problem with multiple uses of vehicles
VRPMC	Vehicle Routing Problem with Multiple Constraints
VRPTW	Vehicle Routing Problem with Time Windows

1 INRODUCTION

1.1 Overview

Supply Chain Management could be defined as the practice of analyzing all aspects of acquiring, storing, moving, and delivering materials from the time they are acquired through any conversion or production processes through to the time final products are used or sold. A company's supply chain may consists of geographically dispersed facilities where raw materials, intermediate products, or finished products are acquired, transformed, stored, or sold, and transportation links connecting the facilities along which products flow.

Logistics management is that part of the supply chain which plans, implements and controls the efficient, effective forward and reverse flow and storage of goods, services and related information between the point of origin and the point of consumption in order to meet customers' requirements. Depending on the industry sector, supply chain logistics costs account from 5% to 50% of a product's total landed cost. The Vehicle Routing Problem (VRP) is an important problem occurring in many distribution systems.

Many companies are faced with problems regarding the transportation of people, goods or information. This is commonly denoted as routing problems. Indeed they not only model the problems of collection and delivery of goods, but, more generally, appear as a key ingredient in many transportation systems, such as those for solid waste collection, street cleaning, bus routing, dial-a-ride systems, routing of maintenance units, transports for handicapped. Another area in which very similar problems play a relevant role is modern telecommunication networks, even if here we find "routing" and not "vehicle routing" problems. As the world economy turns more and more global, transportation is becoming more important. And with the current energy and economic crisis, conserving resources is at the utmost priority.

Environmental Accounts published by the Office for National Statistics in the UK show that, on a UK resident's basis, greenhouse gas emissions fell 1.4 per cent between 2005 and 2006 to 724.5 million tonnes of CO2 equivalent(Office of National Statistics, News Release, 2006). Between 2005 and 2006 greenhouse gas emissions from the non-household sector decreased

by 1.1 per cent to 572.8 million tonnes of CO2 equivalent. This was largely driven by a fall in emissions from the transport and communications sector due to changes in the structure of the UK shipping industry. If shipping industry emissions are removed from the data the year on year change in emissions from the non-household sector rose 0.2 per cent.

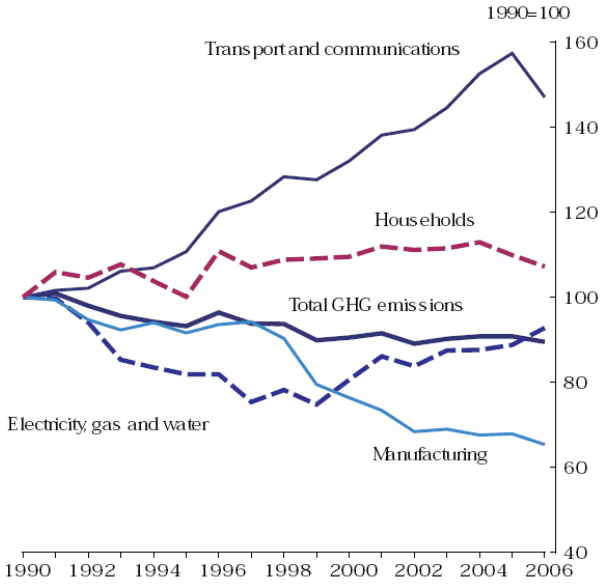

Figure 1 Greenhouse gas emissions in UK

Greenhouse gas emissions from the non-household sector accounted for 79.1 percent of all emissions in 2006. The transport and communications industries were one of the most significant non-household contributors to greenhouse gas emissions in 2006, responsible for 15.7 per cent (113.8 million tonnes of CO2 equivalent) and 13.3 per cent (96.3 million tonnes of CO2 equivalent) respectively. Emissions from the road transport industry show a small year on year increase of 0.4 per cent but at 190.9 million tonnes of CO2 equivalent this is 17.9 per cent above the 1990 level.

In practice, vehicle routing may be the single biggest success story in operations research. For example, each day 103,500 drivers at UPS follow computer-generated routes. The drivers visit 7.9 million customers and handle an average of 15.6 million packages.

Solving different kinds of the Vehicle Routing Problem is an important area of Operations Research. Achieving improvement of only a small percentage may result in large savings and reduce the strain on the environment caused by pollution and noise. The cost of

implementing a solution requires analysts and developers who understand the specific problem for the company. This initial cost of implementation is still a major drawback for companies to take the step towards implementing a solution. It is also viewed that the business is dynamic and parameters might change, which would render the current implementation inefficient. This will cause the company to reinvest into a new solution again.

Smaller companies are not even in the position to consider such an investment. Smaller companies also tend to be more flexible in their service, which let to their problems not being, tied a specific type of method for the solution. The approach of *good-enough* results in better solutions than currently implemented for these types of problems.

We consider the Vehicle Routing Problem in which a fleet of vehicles must service known customer demands for a single commodity from a common depot at minimum cost. This difficult combinatorial problem contains the Travelling Salesman Problem as special case. The problem can consist of multiple constraints such as heterogeneous fleet, multiple time windows and peak and off-peak travel times. In a *pure* routing problem there is only a geographic component, which can be represented by a graph on a two dimensional space. In more realistic routing problems, scheduling plays a major role. Scheduling is represented by the time component.

This study will create a solution for the VRP and some variants with the help of evolutionary algorithms implemented in parallel and build on an adaptive object model approach. The challenge is to combine these methods into one method to find a *good-enough* solution in *acceptable* time. The problems in research are often more simplistic than real-life problems. This research model the approach of flexible methods which would let to an adaptive implementation. A number of important basic models exist and is used as representative problem to assist in the investigation.

The rest of this chapter will discuss the some background of these methods in more detail.

1.2 Background on VRP

One of the most significant problems of supply chain management is the distribution of products between locations, most known as the Vehicle Routing Problem (VRP). The vehicle routing problem is one of the most challenging problems in the field of combinatorial

optimization. Dantzig and Ramser first introduced the VRP in 1959. In a short paper published in Management Science in October 1959 they wrote:

This paper is concerned with the optimum routing of a fleet of gasoline delivery trucks between a bulk terminal and a large number of service stations supplied by the terminal. (Omitted) A procedure based on a linear programming formulation is given for obtaining a near optimal solution. The calculations may be readily performed by hand or by an automatic digital computing machine. No practical applications of the method have been made as yet. A number of trial problems have been calculated, however. (Dantzig and Ramser, 1959)

They proposed the first mathematical programming formulation.

There has been since then a steady evolution in the design of solution methodologies, both exact and approximate, for this problem. In 1964 Clarke and Wright proposed an effective greedy heuristic that improved Dantzig and Ramser approach. Since then, hundreds of models and algorithms were proposed for the optimal and approximate solution of the different versions of the VRP. Vehicle Routing Problems are amongst the most important Combinatorial Optimization problems, because of their difficulty as well as their practical relevance. Yet, no known exact algorithm is capable of consistently solving to optimality instances involving more than 50 customers and often requires relative few side constraints.

Since the Dantzig and Ramser paper appeared, work in the field has exploded dramatically. Today, Google Scholar search of the words 'Vehicle Routing Problem' yields more than 24,800 entries. The June 2006 issue of OR/MS Today provided a survey of 17 vendors of commercial routing software, whose packages are capable of solving average-size problems with 1,000 stops, 50 routes, and two-hour hard-time windows in an execution time of 2 to 10 minutes. (Golden, Raghavan and Wasil, 2008)

While much has been documented about the VRP in major studies that have appeared from 1971 (starting with *Distribution Management* by Eilon, Watson-Gandy, and Christofides) to 2002 (ending with The Vehicle Routing Problem by Toth and Vigo), there are important advances and new challenges that has been raised in the last 5 years or so due to technological innovations such as Global Positioning Systems (GPS), Radio Frequency Identification (RF ID), and parallel computing. The portfolio of techniques for modelling and solving the standard, capacitated VRP and its many variants has advanced significantly. Researchers and

practitioners have developed faster, more accurate solution algorithms and better models that give them the ability to solve large-scale problems.

There are several main survey papers on the subject of VRP's (Toth and Vigo, 2001). A classification scheme was given by Desrochers, Lenstra and Savelsbergh (1990). Valle et al (2009) implements an Integer Programming Formulation, a Branch-and-cut method and a Local Branching to solve a non-capacitated Vehicle Routing Problem. Kallehauge (2008) review the exact algorithms proposed in the last three decades for the solution of the vehicle routing problem with time windows (VRPTW). The exact algorithms for the VRPTW are in many aspects inherited from work on the travelling salesman problem (TSP).

Yeun et al (2008) provides an overview of the methods used in solving the classical VRP, the Capacitated VRP, the VRP with Time Windows and the VRP with Pickup and Delivery. From his study, it is clear the methods used as old as 1995 is still valid in solving the problem today.

Besides being one of the most important problems of operations research in practical terms, the vehicle routing problem is also one of the most difficult problems to solve. The problem is to design routes for the vehicles so as to meet the given constraints and objectives minimizing a given objective function. VRP is a generalization of the travelling salesman problem (TSP), therefore is NP-Hard. The VRP has a finite number of feasible solutions. The VRP solution space increase exponentially as the number of customers increases. The travelling salesman problem is the VRP with one vehicle with no limits, no depot (or any depot), customers with no demand.

In the m-TSP problem, m salesmen have to cover the cities given. Each city must be visited by exactly one salesman. All salesmen start from the same city (the depot) and must end their journey in this city again. We now want to minimize the sum of distances of the routes. The VRP is the m-TSP where a demand is associated with each city, and each salesmen/vehicles has a certain capacity (not necessarily identical). The sum of demands on a route cannot exceed the capacity of the vehicle assigned to this route. As in the m-TSP we want to minimize the sum of distances of the routes. Note that the VRP is not purely geographic since the demand may be constraining.

1.2.1 VRP variants

There exist a number of VRP generalizations.

- CVRP – Capacitated Vehicle Routing Problem: The vehicles have limited carrying capacity of the goods that must be delivered.
- VRPTW - Vehicle Routing Problem with Time Windows: The delivery locations have time windows within which the deliveries (or visits) must be made.
- VRPLC - Vehicle Routing Problem Length Constraint: The routes are limited to a specific length.
- VRPBTW - Vehicle Routing Problem with Backhauling and Time Windows: Backhauling is part of the problem.
- MCVRPTW – Multi Compartment Vehicle Routing Problem with Time Windows: Multiple commodities allowed on the vehicle.
- MDVRPTW – Multi Depot Vehicle Routing Problem with Time Windows: Stops served from more than one depot.
- VRPPD - Vehicle Routing Problem with Pickup and Delivery: A number of goods need to be moved from certain pickup locations to other delivery locations. The goal is to find optimal routes for a fleet of vehicles to visit the pickup and drop-off locations.
- Vehicle Routing Problem with LIFO: In this context LIFO stands for Last In First Out. Similar to the VRPPD, except an additional restriction is placed on the loading of the vehicles: at any delivery location, the item being delivered must be the item most recently picked up. This scheme reduces the loading and unloading times at delivery locations because there is never any need to temporarily unload items to get to the items needing to be dropped off.

This study will focus on handling several variations of the vehicle routing problem through the implementation of constraint functions in the adaptive object model framework. The purpose is to provide an adaptive solution that is *good enough* for most variations. The current computer processor ability allows the research to implement new methods that was previously deemed as too slow.

1.3 Input Data

Solving the vehicle routing problem is a complex task which results in time consuming algorithms. Knowledge of the problem environment can assist in developing more effective algorithms. The problem environment consists of the constraints imposed on the problem, the input data and the objective function to minimize with. In the field of information systems it is customary to distinguish between data, information, and knowledge.

- Data. The term data refers to numeric (or alphanumeric) strings that by themselves do not have a meaning. They can be facts of figures to be processed.
- Information. Information is data organized so that it is meaningful to the person receiving it.
- Knowledge. Knowledge has several definitions. For example, according to the Webster's New Dictionary of the American Language, Knowledge is: *a clear and certain perception of something, understanding, learning, all that has been perceived or grasped by the mind, practical experience, skill, acquaintance or familiarity, organized information applicable to problem solving.*

Data, information, and knowledge can be classified by their degree of abstraction and by their quantity. Knowledge is the most abstract and exists in smallest quantity.

Another definition of knowledge is that given by John F Sowa (Sowa, 2000): "Knowledge encompasses the implicit and explicit restrictions, placed upon objects (entities), operations, and relationships along with general and specific heuristics and inference procedures involved in the situation being modelled."

The Solomon datasets has been used as benchmark for algorithms since it has been published in 1987. Solomon generated six sets of problems. Their design highlights several factors that affect the behaviour of routing and scheduling algorithms. They are:

- Cost relation between customers represented by the geographical location;
- the number of customers serviced by a vehicle;
- percent of time-constrained customers;
- tightness and positioning of the time windows.

Solomon's geographical data are

- randomly generated
- clustered
- mix of random and clustered structures.

Some problem sets have a short scheduling horizon and allow only a few customers per route (approximately 5 to 10). In contrast, other sets have long scheduling horizons permitting many customers (more than 30) to be serviced by the same vehicle.

Current solutions for the VRP use the Solomon datasets for benchmark only. It then concludes that the algorithm was effective on certain problem types. These solutions do not attempt to utilize knowledge of the problem environment to improve results, either for speed of convergence or quality of the answer.

The other approach is to develop an algorithm to solve a specific problem type only. This method can be seen as utilizing knowledge of the problem environment in a static way. These algorithms are generally quick and effective. We are looking for the same efficiency, without the restriction of knowledge on the type of problem build into the algorithm. Methods have been proposed to identify the environment and then select the appropriate algorithm to solve the specific problem. Fuzzy clustering is a preferred way of identifying the problem space. This research will use a similar approach and will utilize the knowledge extensively.

The research will also monitor the behaviour of the operations on the data environment to implement more efficient improvement move combinations. This must be done dynamically, as the aim is still to provide one algorithm that can adapt to the problem environment.

1.4 Algorithms

The optimization of VRP type problems requires us to obtain the least of some measure, namely cost. The problem is so complex that exact algorithms do not have the power to provide high quality solutions to these types of problems. Heuristic methods will be used to assist in solving the problem.

1.4.1 Artificial Intelligence

This past few years have witnessed an increased interest in applied AI. The topic is enjoying tremendous publicity under multiple titles. Many major periodicals have published cover stories on AI or have dedicated special issues to it. Dozens of books on AI have appeared on the market. Many AI newsletters are being published regularly, and conferences and conventions on this topic are being held worldwide. To a certain extent, AI has become a sensation.

The commercial applications of AI are projected to reach several billion dollars annually. Major management consulting firms are deeply involved in applied AI. Many contribute in AI research projects.

These developments may have a significant impact on many organizations, both private and public, and on the manner in which organizations are being managed.

Artificial intelligence is a term that encompassed many definitions. Most experts agree that AI is concerned with two basic ideas. First, it involves studying the thought processes of humans (to understand what intelligence is); second, it deals with representing those processes via machines (computers, robots, etc.)

One well-publicized definition of AI is as follows: Artificial intelligence is behaviour by a machine that, if performed by a human being, would be called intelligent. A thought-provoking definition is provided by Elaine Rich (1983): "Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better." Mark Fox of Carnegie-Mellon University often says that AI is basically a theory of how the human mind works (Turban and Aronson, 2001). Winston and Prendergast (1984) list three objectives of artificial intelligence:

- Make machines smarter (primary goal)
- Understand what intelligence is (the Noble laureate purpose)
- Make machines more useful (the entrepreneurial purpose)

Let us explore the meaning of the term intelligent behaviour. Several abilities are considered signs of intelligence:

- Learn or understand from experience
- Make sense out of ambiguous or contradictory messages
- Respond quickly and successfully to a new situation (different responses, flexibility)
- Use reason in solving problems and directing conduct effectively
- Deal with perplexing situations
- Understand and infer in ordinary, rational ways
- Apply knowledge to manipulate the environment
- Acquire and apply knowledge
- Think and reason
- Recognize the relative importance of different elements in a situation

Although AI's ultimate goal is to build machines that will mimic human intelligence, the capabilities of current commercial AI products are far from exhibiting any significant success when compared with the abilities just listed. Nevertheless, AI programs are getting better all the time, any they are currently useful in conducting several tasks that require some human intelligence.

An interesting test designed to determine if a computer exhibits intelligent behaviour was designed by Alan Turing (1950) and is called the Turing Test. According to this test, a computer could be considered to be smart only when a human interviewer, conversing with both an unseen human being and an unseen computer, could not determine which is which.

The potential value of artificial intelligence can be better understood by contrasting it with natural, or human, intelligence. According to Kaplan (1984), AI has several important commercial advantages:

- AI is more permanent. Natural intelligence is perishable from a commercial standpoint in that workers can change their place of employment or forget information. AI, however, is permanent as long as the computer systems and programs remain unchanged.

- AI offers ease of duplication and dissemination. Transferring a body of knowledge from one person to another usually requires a lengthy process of apprenticeship; even so, expertise can never be duplicated completely. However, when knowledge is embodied in a computer system, it can be copied from that computer and easily moved to another computer, sometimes across the globe.
- AI can be less expensive than natural intelligence. There are many circumstances in which buying computer services costs less than having corresponding human power carry out the same tasks (over the long run)
- AI being a computer technology is consistent and thorough. Natural intelligence is erratic because people are erratic; they do not perform consistently.
- AI can be documented. Decisions made by a computer can be easily documented by tracing the activities of the system. Natural intelligence is difficult to reproduce; for example, a person may reach a conclusion but at some later date may be unable to re-create the reasoning process that led to that conclusion or to even recall the assumptions that were a part of the decision.

Natural intelligence does have several advantages over AI:

- Natural intelligence is creative, whereas AI is rather uninspired. The ability to acquire knowledge is inherent in human beings, but with AI, tailored knowledge must be built into a carefully constructed system.
- Natural intelligence enables people to benefit from and use sensory experience directly, whereas most AI systems must work with symbolic input.
- Perhaps most important, human reasoning is able to make use at all times of a wide context of experience and bring that to bear on individual problems; in contrast, AI systems typically gain their power by having a very narrow focus.

The advantages of natural intelligence over AI result in the many limitations of expert systems.

The definitions of AI presented to this point concentrated on the notion of intelligence. The following definitions and characteristics of AI focus on decision making and problem solving.

1.4.1.1 Symbolic Processing

When human experts solve problems, particularly the type that are considered appropriate for AI, they do not do it by solving sets of equations or performing other laborious mathematical computations. Instead, they choose symbols to represent the problem concepts and apply various strategies and rules to manipulate these concepts. According to Waterman, the AI approach represents knowledge as sets of symbols that stand for problem concepts. In AI jargon a symbol is a string of characters that stands for some real-world concept.

1.4.1.2 Heuristics

Heuristics (rules of thumb) are included as a key element of AI in the following definition: "Artificial intelligence is the branch of computer science that deals with ways of representing knowledge using symbols rather than numbers and with rules-of-thumb, or heuristics, methods for processing information" (Encyclopaedia Britannica)

People frequently use heuristics, consciously or otherwise, to make decisions. By using heuristics one does not have to rethink completely what to do every time a similar problem is encountered. The topic of heuristics will be revisited.

1.4.1.3 Inferencing

Artificial intelligence involves an attempt by machines to exhibit reasoning capabilities. The reasoning consists of inferencing from facts and rules using heuristics of other search approaches. Artificial intelligence is unique in that it makes inferences by employing the pattern-matching (or recognition) approach.

1.4.1.4 Pattern Matching

The following definition of AI focuses on pattern-matching techniques: Artificial intelligence works with pattern-matching methods which attempt to describe objects, events, or processes in terms of their qualitative features and logical and computational relationships.

1.4.2 Heuristic

A heuristic is a replicable method or approach for directing one's attention in learning, discovery, or problem-solving. It is originally derived from the Greek "heurisko" (εὑρίσκω), which means "I find". (A form of the same verb is found in Archimedes' famous exclamation

"eureka!" – "I have found [it]!") The term was introduced in the 4th century AD by Pappus of Alexandria.

Heuristics is the development of methods and rules for the construction of theories and theorems on a non-deductive basis (as opposed to algorithms which provide deductive foundations for such constructions). The heuristic method may be understood as a special case of the trial and error method, i.e. random attempts until a solution is found. There is no secure way. When a solution is found it may, however, be tested with scientific rigor and its truth or falsity may be established.

The heuristic method is different from the deductive method in its application of assumptions, analogies, working hypothesis, and different kinds of models. Heuristics is different from "trial and error" by not using arbitrary assumptions but apply a qualified basis from concepts, models and hypotheses.

Meta-heuristics provided a way of considerably improving the performance of simple heuristic procedures, such as those based on hill climbing. The search strategies proposed by meta-heuristic methodologies result in iterative procedures with the ability to escape local optimal points. Meta-heuristics have been developed to solve complex optimization problems in many areas, with combinatorial optimization being one of the most fruitful. Generally, the best procedures achieve their efficiencies by relying on context information. The solution method can be viewed as the result of adapting meta-heuristic strategies to specific optimization problems.

The term meta-heuristic (also written metaheuristic) was coined by Fred Glover in 1986 (Glover, 1986) and has come to be widely applied in the literature, both in the titles of comparative studies and in the titles of volumes of collected research papers. Meta (from Greek: μετά = "after", "beyond", "with", "adjacent"), is a prefix used in English in order to indicate a concept which is an abstraction from another concept, used to complete or add to the latter. In epistemology, the prefix meta- is used to mean *about* (its own category). For example, metadata are data about data (who has produced them, when, what format the data are in and so on).

A meta-heuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality.

The heuristics guided by such a meta-strategy may be high level procedures or may embody nothing more than a description of available moves for transforming one solution into another, together with an associated evaluation rule.

The evolution of meta-heuristics during the past ten years has taken an explosive upturn. Meta-heuristics in their modern forms are based on a variety of interpretations of what constitutes "intelligent" search. These interpretations lead to design choices that in turn can be used for classification purposes. However, a rigorous classification of different meta-heuristics is a difficult and risky enterprise, because the leading advocates of alternative methods often differ among themselves about the essential nature of the methods they espouse.

This thesis will implement a specific design meta-heuristic algorithm to improve solutions in the current solution space. It will be adapted to perform efficiently in a parallel environment.

1.5 Parallel Algorithms

In computer science, a parallel algorithm, as opposed to a traditional serial algorithm, is one which can be executed a piece at a time on many different processing devices, and then put back together again at the end to get the correct result. Parallel programming was once the sole concern of extreme programmers worried about huge supercomputing problems. With the emergence of multi-core processors for mainstream applications, however, parallel programming is well poised to become a technique every professional software developer must know and master.

Parallel algorithms are valuable because it is faster to perform large computing tasks via a parallel algorithm than it is via a serial (non-parallel) algorithm, because of the way modern processors work. It is far more difficult to construct a computer with a single fast processor than one with many slow processors with the same throughput. There are also certain theoretical limits to the potential speed of serial processors. On the other hand, every parallel algorithm has a serial part and so parallel algorithms have a saturation point (see Amdahl's law). After that point adding more processors does not yield any more throughput but only increases the overhead and cost.

The cost or complexity of serial algorithms is estimated in terms of the space (memory) and time (processor cycles) that they take. Parallel algorithms need to optimize one more resource,

the communication between different processors. Two ways parallel processors communicate is shared memory or message passing.

A multi-agent system (MAS) is a system composed of multiple interacting *intelligent* agents. Multi-agent systems can be used to solve problems which are difficult or impossible for an individual agent or monolithic system to solve. An intelligent agent (IA) is an autonomous entity which observes and acts upon an environment (i.e. it is an agent) and directs its activity towards achieving goals (i.e. it is rational). Intelligent agents may also learn or use knowledge to achieve their goals. The evolutionary approach through Ant Systems relate to a multi-agent system.

The agents in a multi-agent system have several important characteristics:

- Autonomy: the agents are at least partially autonomous
- Local views: no agent has a full global view of the system, or the system is too complex for an agent to make practical use of such knowledge
- Decentralization: there is no designated controlling agent (or the system is effectively reduced to a monolithic system)

Multi-agent systems can manifest self-organization and complex behaviours even when the individual strategies of all their agents are simple. The relation of this multi-agent system approach to the parallel computing resides in the inter-communication between the agents.

Parallel algorithm design is an interesting and challenging area of computer science which requires a combination of creative and analytical skills. It is important to add the discussion to the study, to assure that there exists at least a limited design for parallel implementation. This ease the adaption of the solution for more advanced implementations in the future.

1.6 Adaptive Object Modelling

An adaptive object model is effectively a software factory with an interpretive run time, instead of code generation. It is an object model where the domain representation is interpreted at runtime and can be altered or changed with immediate effect. The adaptive model defines mechanisms to describe entities, attributes and relationships, as well as mechanisms to interpret the domain model and execute business rules. In an ever-changing

business environment, business models and rules have migrated from compiled source code to external metadata. This paradigm empowers domain experts to take control over application implementations, and allows them to change an application's business model as the business evolves.

Data themselves become more universal and reusable when they are accompanied by descriptions of themselves that let other programs make sense of them. They can become even more independent when they are accompanied in their travels by code.

As this evolutionary process unfolds, and the architecture of a system matures, knowledge about the domain becomes embodied more and more by the relationships among the objects that model the domain, and less and less by logic hardwired into the code. Objects in such an active object-model are subject to runtime configuration and manipulation like any other data. Changes to this runtime constellation of objects constitute changes to the model, and to the operations that traverse or interpret it.

Data that describe other data, rather than aspects of the application domain itself, are called metadata. Metadata is when you know something is going to vary in a predictable way and you store the descriptions of the variation in a database so that it is easy to change. The key is to define the problem domain, and then developing both design time and run time variables for that domain to facilitate the development, deployment, execution, operation and maintenance of solutions.

We will implement the concept of adaptive object modelling in describing the problem objects in metadata, as well as interpreting the object model at run-time for assisting the heuristic algorithm in having domain knowledge. We implement the solution through the Expandable Software Infrastructure (ESI) developed by E-Logics (Pty) Ltd: The ESI's goal is to realize runtime configurability, adaptability, extendibility and intuitive configuration requirements through the use of metadata and can be briefly described as a metadata-driven component based framework.

1.7 Summary

Researching the VRP provides an excellent basis to implement new methodologies on various levels. The purpose of this research is to extend the circle of research objectives by proposing a not so traditional objective in the VRP. The problem definition includes additional intentions to be solved.

The design of a solution for the defined problem environment requires a combination of different approaches on different levels. The methods discussed in this thesis are by no mean exhausted, but define the selected methods which this study will utilize. Current solutions rely mostly on heuristics to solve the complex Vehicle Routing Problem and only a few instances utilized the power of parallel algorithms.

Building the solution on an adaptive object model emphasize the importance to structure the problem in more than one dimension. This study focus mainly in providing a solution which is capable of handling adaptive objects. The remaining parts of the thesis formulate the problem, discuss some of the history of the problem and design an approach required to solve the proposed problem.

2 VRP: AN OVERVIEW

2.1 Introduction

This chapter outlines some of the related work that was done on the VRP during the past years. It will also focus on the specific methods used to solve the VRP in its different formats and data environments.

The vehicle routing problem is one of the most challenging problems in the field of combinatorial optimization. Dantzig and Ramser first introduced the VRP in 1959 (Dantzig and Ramser, 1959). The VRP originated from the Travelling Salesman Problem (TSP). The majority of OR oriented minds had been presented with the TSP or variations of, for a very considerable time (Cummings, 2000).

Both the VRP and the TSP are concerned with determination of routes in a graph such that a certain cost associated between nodes is minimized. In fact, if there is only one vehicle with infinite capacity, the consequent VRP can be seen as a simple TSP.

2.2 Travelling Salesman Problem

Mathematical problems related to the travelling salesman problem were treated in the 1800s by the Irish mathematician Sir William Rowan Hamilton and by the British mathematician Thomas Penyngton Kirkman (History of the TSP, 2007). Hamilton created the Icosian game in which the player must find paths and circuits on the dodecahedral graph, satisfying certain conditions. e.g., adjacency conditions, etc. The rights were sold for £25 to a wholesaler dealer in games and puzzles.

The general form of the TSP appears to be have been first studied by mathematicians starting in the 1930s by Karl Menger in Vienna and Harvard. In 1932 Menger published "Das botenproblem", in Ergebnisse eines Mathematischen Kolloquiums (Cummings, 2000). Menger called it the "Messenger Problem" a problem encountered by postal messengers, as well as by many travellers. He went on to define the problem as: "the task of finding, for a finite number of points whose pair wise distances are known, the shortest path connecting

the points. The rule, that one should first go from the starting point to the point nearest, etc., does not in general result in the shortest path."

During the 1950s a number of solutions appeared from the likes of George B. Dantzig, Fulkerson, and Johnson (1954). Their approach remains the only known tool for solving nontrivial TSP instances with more than several hundred cities; over the years, it has evolved further through the work of M.Grtschel, S. Hong, M. Jnger, P. Miliotis, D. Naddef, M. Padberg, W.R. Pulleyblank, G. Reinelt, and G. Rinaldi. George B. Dantzig is generally regarded as one of the three founders of linear programming, along with von Neumann and Kantorovich. (Cummings, 2000)

In 1959 George.B. Dantzig, D.R. Fulkerson, and S.M. Johnson published "On a linear-programming, combinatorial approach to the travelling-salesman problem", Operations Research 7, 59-66 (Dantzig, Fulkerson and Johnson, 1959). This provided a step-by-step application of the Dantzig-Fulkerson-Johnson for a ten city example. This was the same year that Dantzig and Ramser published their first article about the VRP.

2.3 Background on VRP

In the past four decades, a tremendous amount of work in the field of vehicle routing and scheduling problems has been published. They are summarized in recent books and surveys, see Bramel and Simchi-Levi (1997), Braysy et al. (2004), Choi and Tcha (2007), Crainic and Laporte (1998), Desrosiers et al. (1995), Fisher (1995), Laporte (1992) and Nagy and Salhi (2007) (Yeun et al., 2008). Some research efforts were oriented towards the development and analysis of approximate heuristic techniques capable of solving real-size VRP problems. Bowerman, Calamiand and Brent Hall (1994) classified the heuristic approaches to the VRP into five classes:

1. cluster-first/route-second,
2. route-first/cluster-second,
3. savings/insertion,
4. improvement/exchange and

simpler mathematical programming representations through relaxing some constraints.

From the two clustering procedures, the cluster-first/route-second looks more effective. This algorithm first groups the nodes into clusters, assigns each cluster to a different vehicle and, finally, finds the vehicle tour by solving the corresponding travelling salesman problem (TSP). Heuristic methods 3 and 4 permit to construct an initial solution or improve the current set of tours by either inserting customers or exchanging arcs. Some approximate approaches called meta-heuristics, including simulated annealing, tabu search and genetic algorithms, have recently become very popular (Gendrau, Laporte and Potvin, 1997).

On the other hand, effective optimal approaches for VRPTW problems of smaller size have also been reported. Exact approaches can be categorized according to the underlying methodology into: (a) dynamic programming techniques (Kolen, Rinnooy and Trienekens, 1987), which are extensions of the state-space relaxation method of Christofides, Mingozzi and Toth (1981); (b) Lagrangian relaxation methods which are currently capable of optimally solving some 100-customer VRPTW problems, (Desrosiers, Sauve and Soumis, 1988) (Halse, 1992) (Jornsten, Madsen and Sorensen, 1986); (c) column generation algorithms that are based on a combination of linear programming relaxed set covering and column generation (Desrochers, Desrosiers and Solomon, 1992), and (d) K-tree approaches that extended the classical 1-tree method for the TSP to the case with vehicle capacity and time window constraints (Fisher, 1994)(Fisher, Jornsten and Madsen, 1997). The first three exact approaches rely on the solution of a shortest path problem with time windows and vehicle capacity constraints either as part of a Lagrangian relaxation or to generate new columns. (Dondo and Cerdá, 2006)

The past years, quite good results have been achieved for the Vehicle Routing Problem with Time Windows (VRPTW), in both the classes of exact methods and meta-heuristics. Surveys can be found in Toth and Vigo (2001: Chapter 7) and Yeun et al. (2008). Bräysy and Gendreau (2005) give an excellent overview over meta-heuristics for the VRPTW. All of these considered traditionally vehicle routing for which each customer is visited exactly once.

The VRP is an important combinatorial optimization problem. It also occupies a central place in distribution management. Toth and Vigo (2001) report that the use of computerized methods in distribution processes often results in savings ranging from 5% to 20% in transportation costs. Baker and Ayechew (2003) and Laporte (2007) describe several case studies where the application of VRP algorithms has led to substantial cost savings.

The VRP was introduced by Dantzig and Ramser (1959) more than five decades ago. There has been since then a steady evolution in the design of solution methodologies, both exact and approximate, for this problem. Yet, no known exact algorithm is capable of consistently solving to optimality instances involving more than 50 customers (Golden et al., 1998).

Heuristics are usually used in practice. Heuristics include constructive heuristics e.g., Clarke and Wright (1964), which gradually build a feasible solution while attempting to keep solution cost as low as possible, two-phase heuristics on which customers are first clustered into feasible routes and actual routes are then constructed e.g. Fisher and Jaikumar (1981); Gillett and Miller (1974), and improvement methods which either act on single routes by application of a Travelling Salesman Problem (TSP) heuristic, or on several routes by performing customer reallocations or exchanges e.g. Kinderwater and Savelsbergh (1997), Thompson and Psaraftis (1993).

As reported by Laporte and Semet (2002), classical heuristics usually have a low execution speed but often produce solutions having a large gap with respect to the best known (typically between 3% and 7%). In the last fifteen years, several meta-heuristics have been put forward for the solution of the VRP. These typically perform a thorough exploration of the solution space, allowing deteriorating and even infeasible intermediate solutions.

A number of methods maintain a pool of good solutions which are recombined to produce even better ones. There exist many families of meta-heuristics for the VRP: simulated annealing, deterministic annealing, tabu search, genetic algorithms, ant systems, and neural networks. While the success of any particular method is related to its implementation features, it is fair to say that tabu search (TS) is most favoured in the approaches. Extensive computational experiments independently conducted by several researchers corroborate that Tabu Search outperforms most of the competitors on a regular basis. For comparative computational results over the Christofides, Mingozzi, and Toth (CMT) fourteen benchmark instances, see Gendreau, Laporte, and Potvin (Cordeau and Laporte, 2002).

Due to its wide applicability in practical settings, the VRPTW has been an area of intense research during the last ten years. Generally speaking, the methodologies for solving this problem can be classified as:

- Exact algorithms.
- Route construction heuristics.
- Route improvement heuristics.
- Composite heuristics that include both route construction and route improvement procedures.
- Meta-heuristics - like tabu search, simulated annealing, genetic algorithms, evolutionary algorithms and hybrids
- Hyper-heuristics
- Memetic Algorithms

2.3.1 Evolutionary algorithms

Genetic algorithms are adaptive heuristic search methods that mimic evolution through natural selection. They work by combining selection, recombination and mutation operations. The selection pressure drives the population toward better solutions while recombination uses genes of selected parents to produce offspring that will form the next generation. Mutation is used to escape from local minima.

Blanton and Wainwright (1993) were the first to apply a genetic algorithm to VRPTW. They hybridized a genetic algorithm with a greedy heuristic. Under this scheme, the genetic algorithm searches for a good ordering of customers, while the construction of the feasible solution is handled by the greedy heuristic.

Thangiah et al. (1994) test the same approach to solve vehicle routing problems with time deadlines. In the algorithm proposed by Potvin and Bengio (1996), new offspring are created by connecting two route segments from two parent solutions or by replacing the route of the second parent-solution by the route of the first parent-solution. Mutation is then used to reduce the number of routes and to locally optimize the solution. Berger, Salois and Begin

(1998) present a hybrid genetic algorithm based on removing certain customers from their routes and then rescheduling them with well-known route-construction heuristics.

The mutation operators are aimed at reducing the number of routes by rescheduling some customers and at locally reordering customers. Further studies built on the word of Berger through creating new crossover and mutation operators. Berger and Barkaoui (2003) continue to use the genetic algorithm as a base and a scheme was proposed that relies on the concept of simultaneous evolution of two populations pursuing different objectives subject to partial constraint relaxation. The first population evolves individuals to minimize total travelled distance while the second focuses on minimizing temporal constraint violation to generate a feasible solution, both subject to a fixed number of tours.

Homberger and Gehring (1999) propose two evolutionary meta-heuristics based on the class of evolutionary algorithms called Evolution Strategies and three well-known route improvement procedures Or-opt (Or, 1976), λ-interchanges (Osman, 1993) and 2-opt (Potvin and Rousseau, 1995). Gehring and Homberger(2000) use a similar approach with parallel tabu search implementation.

Bräysy, Berger and Barkaoui (2000) hybridize a genetic algorithm with an evolutionary algorithm consisting of several route construction and improvement heuristics. The genetic algorithm by Tan, Lee and Ou (2001) is based on Solomon's (Solomon, 1987) insertion heuristic, λ-interchanges and the well-known PMX-crossover operator. Other studies on various meta-heuristics for VRPTW can be found in Bianchessi and Righini (2007), Berger, Barkaoui and Bräysy (2002) and Yeun et al. (2008)

2.3.2 Initial Solutions

Finding a feasible and integrated initial solution to a hard problem is that the first step in addressing the scheduling issue. Heuristics typically use a greedy approach to obtain a good initial solution in an efficient manner and then incrementally improve the solution by neighbourhood exchange or local searches.

Research illustrates that high quality initial solutions allow meta-heuristics to achieve *higher quality* solutions more quickly. Marius Solomon was one of the first researchers to consider the VRPTW. He designed and analysed a number of algorithms to find initial feasible

solutions for the VRPTW (Solomon, 1987). His sequential insertion heuristic (SIH) gave very good results in most environments, and most current heuristic methods make use of this heuristic (or a variation thereof) to effectively find a feasible starting solution. It has been used frequently: e.g. VRPTW with backhauls, routing heterogeneous vehicles, routing with multiple time windows per customer, dynamic VRPTW, on-line routing, dynamic routing for airport shuttles. (Dullaert and Bräysy, 2003)

Dullaert and Bräysy (2003) introduced a push backward modification on Solomon's SIH. This method recalculates the departure time at the depot when a stop is inserted between the depot and the first stop.

Bianchi and Mastrolilli (2004) argue that because of the close relationship between the VRP and TSP, the fast algorithms for the TSP can be applied in the same manner to solve the VRP. However, the addition of a constraint on the capacity of a vehicle avoids its direct usage. Therefore, when the vehicle's capacity is too low, an optimal TSP tour cannot be anymore optimal for this problem. If the capacity is not so extremely low when compared with the existing demand, an optimal TSP tour can be a good starting solution for applying a heuristic-based algorithm for solving the VRP.

In the case of the VRP with stochastic demand, they observed a higher performance of the meta-heuristics using search strategies typical for solving the TSP, i.e. minimizing total travel distance. This could mean that, due to the stochasticity of the demand, the minimization of constraint violations considering the capacity of the vehicle becomes less important than minimizing its total distance.

Joubert and Claasen (2006) address the shortcomings of Solomon's SIH in that it considers all un-routed nodes when calculating the insertion and selection criteria for each iteration. This method makes it computationally expensive. They introduce a compatibility matrix for identifying and eliminating the obvious infeasible nodes. This results in a more effective and robust route construction heuristic.

2.3.3 VRP and Clustering

The philosophy of cluster first route second has been implemented in various shapes and formats. Clusters of nodes are first defined, then such clusters are assigned to vehicles and sequenced on the related tours and finally the routing and scheduling for each individual tour in terms of the original nodes is separately found.

Dondo and Cerda (2006) published a paper on a cluster-based optimisation approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. They presented a three phased heuristic algorithm approach: phase 1 identified a set of cost-effective feasible clusters, phase two assigns clusters to vehicles and sequences them on each tour by using the cluster-based formulation, phase 3 orders nodes within clusters and scheduling vehicle arrival times at customer locations for each tour.

The solution implements exact elimination rules, which are used to reduce the problem size and thus enhancing the efficiency of the solution algorithm. As in most solutions, these elimination rules are subject to the specific problem environment. Finding a good set of clusters, each one comprising of several customer sites, without relying on routing information is quite a difficult task. This paper introduced a time window-based heuristic algorithm that efficiently assembles customer nodes into a rather low number of feasible clusters.

The heuristic clustering procedures leads to a compact version of the VRPTW. The number of binary variables for a problem with 200 nodes, 10 vehicles and 15 clusters drops from 21 910 to 265, i.e. almost a two order of magnitude reduction. Numerical results indicate that the cluster based optimisation method proved to be quite successful on a variety of Solomon's single depot homogeneous fleet benchmark problems.

2.3.4 Tabu Search

Tabu Search is a memory-based search strategy, originally proposed by Glover (1986), to guide the local search method to continue its search beyond a local optimum. One way of achieving this is to keep track of recent moves or solutions made in the past. Several survey papers and books have been written on Tabu Search (Cordeau and Laporte, 2002).

It was Osman (1993) who proposed the concept of λ-interchanges. In his implementation he uses $\lambda = 2$, thus allowing a mix of single and double vertex moves, and single and double vertex swaps between vehicle routes. Osman tested two strategies for selecting a neighbour solution. In the first, called best admissible (BA), the best non-tabu solution is selected. In the second, called first best admissible (FBA), the first admissible improving solution is selected if one exists; otherwise the best admissible solution is retained. Osman shows through empirical testing that with the same stopping criterion FBA produces slightly better solutions, but this variant is much slower than BA. Osman's Tabu Search implementation uses fixed tabu tenures, no long-term memory mechanism and no intensification schemes.

In Taburoute neighbour solutions are obtained by moving a vertex from its current route *r* to another route *s* containing one of its closest neighbours. Insertion into route *s* is performed concurrently with a local reoptimization. This may result in creating a new route or deleting one. To limit the neighbourhood size, only a randomly selected subset of vertices is considered for reinsertion in other routes.

The Adaptive Memory Procedure (AMP) of Rochat and Taillard (1995) was presented under the title "Probabilistic Diversification and Intensification". If applied periodically during the search process, it provides a diversification process by allowing new high quality solutions to emerge. If applied as a post-optimizer, it is best seen as an intensification procedure. The method should not be regarded as a VRP heuristic per se, but rather as a general procedure applicable to several contexts and in conjunction with several heuristic schemes. For example, it was applied by Bozkaya, Erkut and Laporte (2003) to post optimize political districts obtained by means of a Tabu Search heuristic.

Xu and Kelly (1996) introduced a network flow model as a general local search strategy to solve the VRP. They used a straightforward model by relaxing the hard side constraints and introducing a dynamic penalty system, and efficiently update and frequently solve the network flow model to find the best customers to insert into new routes without the use of the generalized assignment problem. The penalty parameters are changed such that the feasibility of the search is controlled.

Garcia, Potvin and Rousseau (1994) describe a tabu search heuristic where the neighbourhood is restricted to the exchange of arcs that are close in distance. The initial solution is created using Solomon's I1 insertion heuristic, and the algorithm oscillates between

2-opt (Potvin and Rousseau, 1995) and Or-opt (Or, 1976) exchanges. When one has not made any improvement for a certain number of iterations, the other improvement operator is used and vice versa.

In order to minimize the number of routes, the algorithm tries to move customers from routes with a few customers into other routes using Or-opt exchanges. The parallel implementation is performed by partitioning the neighbourhood among slave processors. The master processor is then used to guide the tabu search. After exploration of the neighbourhood, the best move from each processor is sent to the master.

The granularity concept proposed by Toth and Vigo (1998) does not only apply to the VRP or to Tabu Search algorithms, but to discrete optimization on the whole. Like AMP, it is a highly portable mechanism. The idea is to permanently remove from consideration long edges that have only a small likelihood of belonging to an optimal solution.

More specifically, a threshold is defined and the search is performed on the restricted edge set $E(v) = \{(v_i, v_j) \in E : c_{ij} \leq v\} \cup I$, where I is a set of *important* edges defined as that incident to the depot. The value of v is set equal to $\beta\bar{c}$, where β is called a sparsification parameter, and $\bar{c}$ is the average edge cost in a good feasible solution quickly obtained, for example, by the Clarke and Wright (1964) algorithm. In practice, selecting β in the interval [1.0; 2.0] results in the elimination of 80% to 90% of all edges. Toth and Vigo have applied this idea in conjunction with Taburoute (Gendreau, Hertz and Laporte, 1994). These approaches is viewed as forced learning and relate closely to the probability matrix used in this study.

Badeau et al. (1997) study the problem using a 2-level parallel implementation that combines the so-called master-slave scheme with an allocation of each sub problem to a different processor. In this master-slave scheme, the master process manages the adaptive memory and generates solutions from it; these solutions are then transmitted to slave processes that improve them by performing tabu search and return the best solutions found to the master. Results on benchmark problems show that parallelization of the original sequential approach does not degrade solution quality, for the same amount of computation, while providing substantial speed-ups. This parallel implementation creates independent operating agents and utilise the processing power available. It does not contribute to the meta-heuristic, or memory control of the Tabu Search.

Carlton (1995) describes a reactive tabu search that dynamically adjusts its parameter values based on the current search status. More precisely, the size of the tabu list is managed by increasing the tabu list size if identical solutions occur too often and reducing it if no feasible solution can be found. This approach is applied to several types of problems with time windows. Its robustness comes from a simple neighbourhood structure, which can be easily adapted to different problems. Namely, each customer is removed and reinserted at some other location in the current solution.

Schulze and Fahle (1999) propose a tabu search performing several search threads in parallel. Each thread is started with a different initial solution and a neighbouring solution is generated by performing a sequence of simple customer shifts (ejection chain). All routes generated by the tabu search heuristic are collected in a pool. At the termination of local optimization steps, the worst solution is replaced by a new one created by solving the set covering problem on the routes in the pool with Lagrangian relaxation based heuristic.

With this new set of solutions, the whole process is restarted until a certain stopping criterion is fulfilled. In addition, the proposed method tries to eliminate routes having at most three customers by trying to move these customers into other routes. The routing of customers supplied by the same vehicle is improved by performing Or-opt exchanges within the route and the search is diversified by penalizing frequently performed customer shifts.

To generate an appropriate number of initial solutions, three different heuristics are used, namely Solomon's (Solomon, 1987) I1 insertion heuristic, the parallel route building heuristic of Potvin and Rousseau (1993) and a modified version of the Savings heuristic of Clarke and Wright (1964). In the parallel implementation, each processor handles a set of solutions instead of just one and solves also the set covering problem separately on these solutions to avoid idle times. Each time a processor terminates its local optimization process, the routes of the optimized solutions are sent to all other processors to enable sharing of knowledge.

Gehring and Homberger (2000) study a two-phase approach, where the tabu search is combined with evolutionary algorithm ES1. In this evolutionary algorithm the search is mainly driven by mutation based on Or-opt, 2-opt* and λ–interchange moves with $\lambda = 1$. In addition a special Or-opt based operator is used to reduce the number of routes.

The individuals of a starting population are generated by means of a stochastic approach that is based on the savings algorithm of Clarke and Wright (1964). The evolutionary algorithm is used in the first phase to minimize the number of routes. In the second phase, the total distance is minimized using a tabu search algorithm utilizing the same local search operators. The approach is parallelized using the concept of cooperative autonomy, i.e., several autonomous sequential solution procedures cooperate through the exchange of solutions. The cooperating slave processes are configured in different ways using different seeds for random number generators to create diversity in the search. (Gendreau and Bräysy, 2001)

Potvin and Naud (2009) presented a variant on the classical vehicle routing problem, where a customer request for a transportation company can be serviced either by its private fleet of vehicles or assigned to an external common carrier. A tabu search heuristic with a neighbourhood structure based on ejection chains is used to solve the problem. The implementation is based on the computation of a least-cost ejection path in a graph structure. This method allows multiple displacements of customers on vehicles of different types which make it effective on large heterogeneous instances.

Moccia, Cordeau and Laporte (2010) describe an incremental neighbourhood tabu search (ITS) heuristic for the generalized vehicle routing problem with time windows. The purpose of this work is to offer a general tool that can successfully be applied to a large variety of specific problems. The algorithm builds upon a previously developed tabu search heuristic by replacing its neighbourhood structure. The new neighbourhood is exponential in size, but the proposed evaluation procedure has polynomial complexity. It uses a shortest path calculation which can be computed by the so-called reaching algorithm in an acyclic graph. The computations are speed up by taking advantage of the shortest paths computed at previous steps of the algorithm.

2.3.5 Ant Optimisation

The Ant System approach, originally proposed by Colorni, Dorigo and Maniezzo (1991) is based on the behaviour of real ants searching for food. Real ants communicate with each other using an aromatic essence called pheromone, which they leave on the paths they traverse. In the absence of pheromone trails ants more or less perform a random walk.

Bullnheimer, Hartl and Strauss (1997) use the Ant System to solve the VRP in its basic form, i.e. with capacity and distance restrictions, one central depot and identical vehicles. A 'hybrid' Ant System is used with specific information for improvement. To solve the VRP, the artificial ants construct vehicle routes by successively choosing cities to visit, until each city has been visited.

Whenever the choice of a city would lead to infeasible solution for reasons of the constraints, the depot is chosen and a new tour started. For the selection of a yet not visited city, two aspects are taken into account: how good was the choice of that city, information stored in the pheromone trial, and how promising is the choice of the city, a measure of desirability.

It was found that the number of ants to start with should be the same as the number of cities in the problem space to allow each ant to start from another city. After initializing the basic ant system algorithm, the two steps, construction of vehicle routes and trial update, are repeated for a given number of iterations.

The 2-opt heuristic for the TSP is used to ensure that each tour is a 2-optimal tour, i.e. there is no possibility to shorten the tour by exchanging 2 arcs. A savings value measures the favourability of combining two cities and calculates their relative location to each other as well as to the depot. Capacity utilization is also measured as a factor to determine the probability of the next city. Although good results were obtained, a tabu search heuristic still outperforms their approach.

Mailleux, Deneubourg and Detrain (2000) compared the behaviour of Lasius niger scouts at sucrose droplets of different volumes, and empirically identified the criterion used by each scout to assess the amount of food available as well as the rules governing its decision to lay a recruitment trail. When scouts discovered food volumes exceeding the capacity of their crop, 90% immediately returned to the nest laying a recruitment trail.

In contrast, when smaller food droplets were offered, several scouts stayed on the foraging area, presumably exploring it for additional food. If unsuccessful, they returned to the nest without laying a trail. The droplet volume determined the percentage of trail-laying ants but had no influence on the intensity of marking when this was initiated. The key criterion that regulated the recruiting behaviour of scouts was their ability to ingest their own desired volume.

This volume acted as a threshold triggering the trail-laying response of foragers. Collective regulation of foraging according to food size resulted from the interplay between the distribution of these desired volume thresholds among colony members and the food volume available. We borrow from the behaviour and note that even the ants have mechanisms to balance between diversification and intensification of the improvement part.

Gambardella, Taillard and Agazzi (1999) presented an Ant Colony Optimization which is organized with a hierarchy of artificial ant colonies designed to successively optimize a multiple objective function: the first colony minimizes the number of vehicles while the second colony minimizes the travelled distances. Cooperation between colonies is performed by exchanging information through pheromone updating. The basic ACO idea is that a large number of simple artificial agents are able to build good solutions to hard combinatorial optimization problems via low-level based communications.

Montemanni et al. (2003) developed a new algorithm for dynamic VRP base on Ant Colony System. The ACS-DVRP algorithm they propose for the DVRP is based on three main elements. First, there is an event manager, which collects new orders and keeps trace of the already served orders and of the current position of each vehicle. The event manager uses this information to construct a sequence of static VRP-like instances, which are solved heuristically by an ACS (Ant Colony System) algorithm, the second element of our architecture.

The third element, the pheromone conservation procedure, is strictly connected with the ACS algorithm. It is used to pass information about characteristics of good solutions from a static VRP to the following one. The Ant Colony System (ACS) algorithm is an element of the Ant Colony Optimization (ACO) family of algorithms. The main underlying idea was to parallelize search over several constructive computational threads. A dynamic memory structure, which incorporates information on the effectiveness of previously obtained results, guides the construction process of each thread. The behaviour of each single agent is inspired by the behaviour of real ants.

Gambardella et al. (2003) present a modular approach to ALS (advanced logistics systems) design and implementation, driven by the user needs. They show how different algorithms and modules can be implemented in an ALS and how tailor-made solutions can be integrated into traditional supply chain management software. An always increasing number of large and

medium-large distribution companies have already adopted ALS to manage their whole supply chain.

The basic information processing infrastructure is in place and many supply chain management suites already provide optimization modules for some components. The objective of their AntRoute software component is to integrate a state-of-the-art optimization algorithm within an existing supply chain management structure. AntRoute has been implemented in C++ and it has been deployed as windows DLL, but its code can be recompiled under most operating systems. The algorithm has been modelled after MACS-VRPTW (Gambardella, Taillard and Agazzi, 1999)

Reimann, Doemer and Hartl (2003) have developed a generalized Ant System. Generally, the Ant System algorithm consists of the iteration of three steps: Generation of solutions by ants according to private and pheromone information, application of a local search to the ants' solutions and update of the pheromone information. They use an Insertion algorithm derived from the I1 insertion algorithm proposed by Solomon for the VRPTW. The algorithm is adapted to allow for a probabilistic choice in each decision step. This is done by choosing seed customers probabilistically according to their distance from the depot.

Inserting further customers on the current tour is done using a roulette wheel selection overall un-routed customers with positive evaluation function κ_i. The chosen customer i is then inserted into the current route at its best feasible insertion position. To compute the evaluation function κ_i for inserting an unrouted customer i at its best insertion position on the current tour we first determine for each un-routed customer i the attractiveness of insertion at any feasible insertion position on the current tour.

Given the attractiveness they then compute the evaluation function of the best insertion position for each customer i on the current tour. After all ants have constructed their solutions, the pheromone trails are updated on the basis of the solutions found by the ants.

Bianchi and Mastrolilli (2004) carried out research which focuses on the Vehicle Routing Problem with stochastic demand, a variation of deterministic classical routing problems, where each customer demand is assumed to follow a given probability distribution, instead of having a single known value. The ACO meta-heuristic is implemented for the deterministic problem in the research.

The stigmergic information is stored in the form of a matrix and at the beginning of the algorithm these values are initialized to a parameter τ_0, except for those elements that belong to the starting solution, generated by the farthest insertion heuristic, who receive a 'reinforcement' equal to *r* iterations of global update rule. After each construction step, a local update rule is applied to the element of the matrix corresponding to the chosen customer pairs.

For the VRP with stochastic demand, the ACO was implemented as two algorithms as follows: ACS-0 where after the a-priori solution is constructed, the OrOpt-0 local search is applied, and ACS-tsp where the OrOpt-tsp local search is used instead of OrOpt-0. The OrOpt-0 and OrOpt-tsp differ in the way the neighbouring solutions are evaluated. The first version exploits the VRP with stochastic demand's objective function, while the second version exploits the TSP objective function. Results show not a significant difference from using the Ant Colony algorithms from other methods for the VRP with stochastic demand.

Despite the common principles of intensification and diversification, meta-heuristics can be profoundly different and also their applicability to a given problem can produce varied results (Rizzoli et al., 2004). Moreover, it was also remarked by Martin, Otto, and Felten that meta-heuristics tend to perform very well when *hybridized* with local search methods, combining specific problem knowledge in the improvement of the solutions. ACO is particularly apt for hybridization, since it is rarely able to build a solution that is good enough, but on the other hand it produces good candidate solutions which can be further improved using various local search techniques.

In their conclusion, after more than ten years of research, ACO has proven to be one of the most successful meta-heuristics and its application to real world problems demonstrates that it has now become a fundamental tool in applied Operational Research and Management Science.

2.3.6 Hyper-Heuristics

A hyper-heuristic is a heuristic search method that seeks to automate, often by the incorporation of machine learning techniques, the process of selecting, combining, generating or adapting several simpler heuristics (or components of such heuristics) to efficiently solve computational search problems. Hyper-heuristics can be thought of as "heuristics to choose

heuristics". One of the motivations for studying hyper-heuristics is to build systems which can handle classes of problems rather than solving just one problem.

The fundamental difference between meta-heuristics and hyper-heuristics is that most implementations of meta-heuristics search within a search space of problem solutions, whereas hyper-heuristics always search within a search space of heuristics. One of the motivations of hyper-heuristic research is to investigate the development of adaptive decision support systems that can be applied to a range of different problems and different problem instances. One possible approach is to dynamically adjust the preferences of a set of simple low-level heuristics (or neighbourhood operators) during the search.

Bai et al. (2007) research a simulated annealing hyper-heuristic technique in order to investigate how the algorithm can intelligently choose between different neighbourhood operators (heuristics) according to the different problems. They consider two of the most popular variants of vehicle routing problems (capacitated VRP and VRP with time windows) and investigate the adaptation of the heuristic-selection mechanism across these variants and at different stages of the search.

Specifically, they investigate: 1). How the hyper-heuristic adapts to these two types of vehicle routing problems by changing preferences of low-level heuristics and whether the hyper-heuristic can automatically identify heuristics that are particularly good for a given type of vehicle routing problem? 2). How the hyper-heuristic adapts its selection decision during different stages of the search when solving a particular problem instance? Their hypothesis is that the hyper-heuristic algorithm will produce good quality solutions across a range of problem instances, without having to resort to tuning parameters for each instance.

Cuesta-Cañada, Garrido and Terashima-Marín (2005) use the idea behind hyper-heuristics which is to find some combination of simple heuristics to solve a problem instead than solving it directly. In this paper they introduce the first attempt to combine hyper-heuristics with an ACO algorithm. The resulting algorithm was applied to the two-dimensional bin packing problem, and encouraging results were obtained when solving classic instances taken from the literature. The performance of our approach is always equal or better than that of any of the simple heuristics studied, and comparable to the best meta-heuristics known.

2.3.7 Memetic Algorithms

Memetic Algorithms (MA) is used as a synergy of evolutionary or any population-based approach with separate individual learning or local improvement procedures for problem search. Cultural evolution, including the evolution of knowledge, can be modelled through the same basic principles of variation and selection that underlie biological evolution. This implies a shift from genes as units of biological information to a new type of units of cultural information: memes.

A meme is a cognitive or behavioural pattern that can be transmitted from one individual to another one. Since the individual who transmitted the meme will continue to carry it, the transmission can be interpreted as a replication: a copy of the meme is made in the memory of another individual, making him or her into a carrier of the meme. (Dawkins, 1976)

Berger and Barkaoui (2002) involves parallel co-evolution of two populations to solve the VRPTW. The first population evolves individuals to minimize total travelled distance while the second focuses on minimizing temporal constraint violation to generate a feasible solution. New genetic operators have been designed to incorporate key concepts emerging from promising techniques such as insertion heuristics, large neighbourhood search and ant colony systems to further diversify and intensify the search.

The parallel version of the method is based on a master-slave message-passing paradigm. The master controls the execution of the algorithm, synchronizes atomic genetic operations and handles parent selection while the slaves concurrently execute genetic operations. Results from a computational experiment show that the serial version of the proposed technique matches or outperforms the best-known heuristic routing procedures. Alternatively, simulation results obtained for the parallel version show a significant improvement over the serial algorithm, matching or even improving solution quality. The parallel algorithm shows a speed-up of five in computing solution having near similar quality.

Prins and Bouchenoua (2002) present a memetic algorithm for solving a new vehicle routing problem that generalizes two classics, the VRP and the CARP. An extended model is introduced: the NEARP (Node, Edge and Arc Routing Problem). It is defined on a mixed graph with required nodes, edges and arcs and contains the VRP and the CARP as particular cases. A common data structure shared by all algorithms is proposed for coding NEARP

instances. The first algorithms developed are three simple heuristics that are used to initialize the initial population of the MA. The third heuristic, a tour splitting method, plays also a key-role in chromosome evaluation.

A memetic algorithm for the NEARP is developed. It manipulates chromosomes corresponding to sequences of tasks, without trip delimiters, allowing adaptations of simple crossovers like OX or LOX. The tour splitting technique designed for the third heuristic is used to split the chromosomes into trips

Mendoza et al. (2010) presents a Multi-Compartment Vehicle Routing Problem (MC-VRP) which consists of designing transportation routes to satisfy the demands of a set of consumers for several products that because of incompatibility constraints must be loaded on independent vehicle compartments. Memetic algorithms are evolutionary algorithms that use local search procedures to intensify the genetic search. The proposed MA shares some of the elements that have been proven effective on the distance-constrained VRP. Starting from an initial population $\mathcal{P}(0)$ comprised of P individuals, the algorithm runs for T generations.

At every generation *t*, crossover, mutation, and local search operators are applied with probabilities p_c, p_m and p_{ls}, respectively. The offspring produced by the operators join the current population to form an expanded population $\mathcal{E}(t)$, from which the best P individuals are selected to become part of the next generation, namely $\mathcal{P}(t+1)$. Clones, which are individuals sharing the same value of the objective function, are completely forbidden in the population to foster diversification in the objective space.

2.4 Problem Overview

There exist ample research and methods to solve the VRP. Solutions can be summarized on two levels: the methodology to follow, e.g. meta-heuristic methods and the strategy implemented e.g. local knowledge of the problem. Almost all research focus on solving a specific case of the VRP in a known environment. The problem in this thesis is unique and existing research provides a toolset to work out an answer for the problem.

It is argued that in order to tackle a complex problem domain, the first thing to do is to construct a well-structured problem formulation, i.e. a "representation". This requires identifying a problem by specifying the undesirable and problematic state currently occupied,

the resources currently available to move away from that problematic state, particularly the available courses of actions, the combinatorial constraints on using them, etc., and the criteria that need to be satisfied to say that a problem no longer exists or is solved.

Problem formulation is the creative and probably the more important step towards overcoming a problematic state than problem-solving. A good definition of what the problem is, is believed to be more than half of the way towards its eventual elimination. (Krippendorff)

The remainder of this chapter identifies and formulates the problem domain as well as the resources used in solving the problem. It first presents the basic interpretation of the Vehicle Routing Problem, followed by an overview of some of the variants. The final sections of the chapter are devoted to methods and scenarios that influence the ease of solving the complex problem. The chapter conclude with the actual problem aimed to be solved.

2.5 Formulation: Vehicle Routing Problem

Logistics can be defined as the provision of goods and services from a supply point to various demand points. The transportation of raw materials from the suppliers to the factory, from the factory to the depots, and the distribution to customers can be described as a complete logistic system. With an effective logistic system, cost can be reduced due to fewer penalties for late delivery, lowered trucking cost, shorter distances and effective use of capacity of the vehicle. One of the most significant measures of a logistic system is effective vehicle routing. Optimizing of routes is the basis of vehicle routing problems.

The VRP originated from the Travelling Salesmen Problem (TSP). According to Winston (Winston, 1994) the TSP can be define as a problem where a salesperson must visit each of ten cities once before returning to his home. The cities need to be selected to minimize the total distance the salesmen travels.

According to Barbarosoglu and Ozgur (1999) the VRP can be described as the problem of designing optimal delivery or collection of routes from one or several depots to a number of customers subject to side constraints. Thus, the basic VRP can be described as vehicles that depart from the depot, visit one or more customers and return to the depot.

The VRP has a finite number of feasible solutions. The solution space increase exponentially as the number of customers increases. Thus the VRP is known as a non-polynomial hard (NP-hard) problem.

The basic VRP is today no more than a classical problem. The advance of science has prompted the industry to ask for more real life solutions. The basic VRP is given by a set of identical vehicles, a depot, a set of customers to be visited and a directed network connecting the depot and customers. Let us assume there are K vehicles, V = {0,1,2,3,…K-1}, and N+1 customers, C = {0,1,2,3,…N}. We denote the depot as customer 0, or C_0. Each arc in the network corresponds to a connection between two nodes. A route is defined as starting from the depot, going through a number of customers and ending at the depot. A cost c_{ij} and a travel time t_{ij} are associated with each arc of the network.

The problem is to find tours for the vehicles in such a way that:

- The objective function is minimized. The objective function can be the total travel distance, the number of vehicles used, or any cost related function.

Several constraints must be applied on the **basic** VRP:

- Only one vehicle handles the deliveries for a given customer. We will not split deliveries across multiple vehicles. A customer can only be visited once a day.
- The number of vehicles is equal to the number of routes, meaning that a vehicle can only complete one route per day.
- The demand of the customers on every route is known with certainty. The demand of the customers in total on one route cannot exceed the capacity of the specific vehicle that will cover that route.
- The travelling distance between customer *i* and *j* are the same as the travel distance between *j* and *i*.
- The vehicles have the same capacity with the same fixed and variable cost, thus a homogeneous fleet is assumed.
- The vehicles must complete their route within a maximum length of time, usually the time the depot is open.

- The vehicle returns to the depot at the end of the route.

The VRP can be formulated as follows:

- A set of identical vehicles V
- A special node called the depot,
- A set of customers C to be visited
- A directed network connecting the depot and the customers

Let us assume there are

K vehicles, V = {0, 1, 2,…, K – 1}, and N + 1 customers, C = {0, 1, 2, … , N}.

For simplicity, we denote the depot as customer 0.

- Each arc in the network corresponds to a connection between two nodes.
- A route is defined as starting from the depot, going to any number of customers and ending at the depot.
- The number of routes in the traffic network is equal to the number of vehicles used, K. Therefore, exactly K directed arcs leave the depot and K arcs return to the depot.
- A cost c_{ij} and a travel time t_{ij} are associated with each arc of the network.
- Every customer in the network must be visited only once by one of the vehicles.
- Since each vehicle has a limited capacity q_k, and each customer has a varying demand m_i, q_k must be greater than or equal to the summation of all demands on the route travelled by vehicle k.
- Vehicles are also supposed to complete their individual routes within a total route time, which is essentially the time window of the depot.

There are two types of decision variables in a VRP.

- The decision variable $x_{ijk,}(i, j = 0,1,2..N; k = 0,1,2..K; i \neq j)$ is 1 if vehicle k travels from node i to node j, and 0 otherwise.

- The decision variable t_i denotes the time a vehicle starts service at node i. The triangular inequality, i.e. $c_{ij} < c_{ih} + c_{hj}$ and $t_{ij} \leq t_{ih} + t_{hj} \forall h, i, j \in N$ need not apply.

The objective is to design a set of cost-minimizing routes that service all the customers while all the constraints stated above are satisfied. The model can be mathematically stated as follows:

Notation:

K = total number of vehicles.	N = total number of customers.
c_i = customer i, where i = 1, 2, … ,N.	c_0 = the depot.
c_{ij} = cost incurred on arc from node i to j.	t_{ij} = travel time between node i and j.
m_i = demand at node i.	q_k = capacity of vehicle k.
e_i = open time at node i.	l_i = close time at node i
t_i = arrival time at node i.	f_i = service time at node i.
r_k = maximum route time allowed for vehicle k.	p_i = polar coordinate angle of c_i.
R_k = vehicle route k.	O_k = total overload for vehicle k.
T_k = total tardiness for vehicle k.	D_k = total travel distance for vehicle k.
W_k = total travel time for vehicle k.	$C(R_k)$ = cost of the route R_k based on cost function.
$C(S)$ = sum total cost of individual routes $C(R_k)$.	

Table 1: Notation

Principle decision variable: $x_{ijk} = \{0,1\}$: 0 if there is no arc between node *i* and *j* and 1 otherwise.

$$Min \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{k=0}^{K-1} c_{ij} x_{ijk} \tag{1}$$

Subject to:

$$\sum_{k=0}^{K-1} \sum_{j=1}^{N} x_{ijk} = K \quad for\ i = 0 \tag{2}$$

$$\sum_{j=1}^{N} x_{ijk} = \sum_{j=1}^{N} x_{jik} \leq 1 \quad for\ i = 0; k \in [0, K-1] \tag{3}$$

$$\sum_{k=0}^{K-1} \sum_{j=1}^{N} x_{ijk} = 1 \quad for\ i = 1, 2..N \tag{4}$$

$$\sum_{i=0, i \neq h}^{N} x_{ihk} - \sum_{j=1, j \neq h}^{N} x_{hjk} = 0 \quad \forall h \in [1, N]; k \in [0, K-1] \tag{5}$$

$$u_i - u_j + N x_{ij} \leq N - 1 \quad for\ i \in [1, N]; j \in [1, N]; i \neq j \tag{6}$$

$$\sum_{i=0}^{N} m_i \sum_{j=0, j \neq i}^{N} x_{ijk} \leq q_k \quad \forall k \in [0, K-1] \tag{7}$$

$$\sum_{i=0}^{N} \sum_{j=0, j \neq i}^{N} x_{ijk} (t_{ij} + f_i + w_i) \leq r_k \quad \forall k \in [0, K-1] \tag{8}$$

- The objective function of the problem is given in (1).
- Constraint (2) specifies that there are exactly K routes going out of the depot.
- The third constraint (3) makes sure that each route leaves the depot and return to the depot
- Constraints (4) and (5) make sure exactly one vehicle goes to and leaves a customer.

- Constraint (6) ensures that there are no sub-tours in the solution. A sub-tour is a route that does not pass through the depot.
- (7) is the capacity constraint.
- Maximum travel time for each vehicle is assured in Eq. (8).

This paragraph describes the elements of the **basic** VRP. These basic principles will be altered to fit into the generic framework designed. The next paragraph investigates some variants on the VRP and the impact on the basic VRP.

2.6 VRP variants

The model described in this section is a standard mathematical model for a basic VRP problem. When additional constraints are needed, they must be added to the existing constraints in the model or some of the existing constraints must be relaxed.

The industry requires additional constraints on the basic VRP. Additional constraints that can be included consist of the following:

- The limitation of the length, duration or cost of each individual tour. This restricts a route for running too long, which can result in overtime costs, insufficient fuel, etc.
- The addition of a variable service time for each customer. The volume of the stock to be delivered can have an influence on the service time at a customer. The delivery time will have an influence on the total route time and must be taken into account.
- The addition of time windows during which the customers have to be visited. The problem we will discuss is the use of multiple time windows, i.e. the customer can specify more than one time period available for delivery.
- The vehicle can return to the depot and have enough time for another route before the maximum allowed time is up. This will allow double scheduling, which will result in a cost saving, as the second route utilize the same vehicle and reduce the number of vehicles required to service all the customers.
- The travel time can vary between customers depending on the time of day. This implies peak and off-peak travel times.

- The fleet is not necessarily homogeneous, i.e. vehicles can differ in capacity and cost. This might result in a good solution to use the vehicles with a large capacity to pick up customers that is far away from the depot.
- A vehicle can have a specified available time. This allows for certain vehicles to be out in the field longer to cater for long routes. The implementation will add time window constraints to a vehicle.

Implementation of these variants requires a redefinition of the mathematical model for our problem, for example if we allow double scheduling:

- Constraint (2) is now invalid and will be replace by

$$\sum_{j=1}^{N} x_{ijk} \le p_k \quad for\ i = 0; k \in [0, K-1] \tag{2}$$

where p_k is the maximum number of routes allowed for vehicle k.

The number of routes going out of the depot for a specific vehicle is constrained to a maximum of p_k, which implies that a vehicle can now have multiple routes done in a day.

- If we impose time windows at a stop

$$t_0 = 0 \tag{9}$$

$$t_i + x_{ijk}(t_{ij} + f_i + w_i) \le t_j \quad i, j \in [1, N]; i \ne j; k \in [0, K-1] \tag{10}$$

$$e_i \le t_i \le l_i \tag{11}$$

- If we redefine the service time at each stop as

f_i = Fixed Time + (Variable Time * m_i)

- If we redefine the meaning of travel time

t_{ij} = Travel Time at $(t_i + f_i + w_i)$

which calculates the travel time from i to j depending on the departure time at i.

- We just make a note that q_k is not necessarily the same for each vehicle.
- The monetary cost of a route can be calculated as follows

$$C(R_{ki}) = (F_k / \sum_{j=1}^{N} x_{ijk}) + (D_k * V_k) \quad for\ i = 0; k \in [0, K-1]$$

where the first term is the fixed cost of the vehicle divided into the number of routes and the second term is the distance of the route multiplied by the running cost of the vehicle.

2.6.1 Multiple Depots

Solving the Vehicle Routing Problem is a complex task. Allowing multiple depots into the problem formulation increase the number of solutions in the problem space exponentially. For this reason, VRP problems are generally viewed form a single depot problem.

The multiple depot concept is important to consider in the context of this study. It provides information on the construction of the problem from the raw input data. The method of handling multiple depots contributes to the approach to handle different groups of customers. The simplistic methodology used, as well as the variables considered, assists with designing the complete solution in this problem environment.

The study follows a divide and conquers approach. The generally accepted method for solving the multiple depots problem utilise the same principle. The first step towards solving the multiple depots VRP is to break it up in solvable problem spaces. Stops are allocated to depots using the following procedure:

- Create a formula that results in a discrete value per stop per depot, e.g. the distance or cost between the stop and the depot.
- Apply this formula on a stop for each depot.
- Get the minimum result and allocate the stop to the depot associated.

This study classifies this allocation method as inefficient if the discrete function does not contain other solution parameters such as neighbouring stops and vehicles. The solution is not dependent on discrete stop information, but is the result of stops interacting with others

stops and vehicles, etc. The formula should result in a discrete value for a stop, but should not only depend on the explicit relations to a depot.

The allocation of a stop to a depot can influence the overall optimization, and careful consideration should be applied in the selection of the formula. The figure below depicts a simple scenario where the allocation of stops has a major impact on the solution.

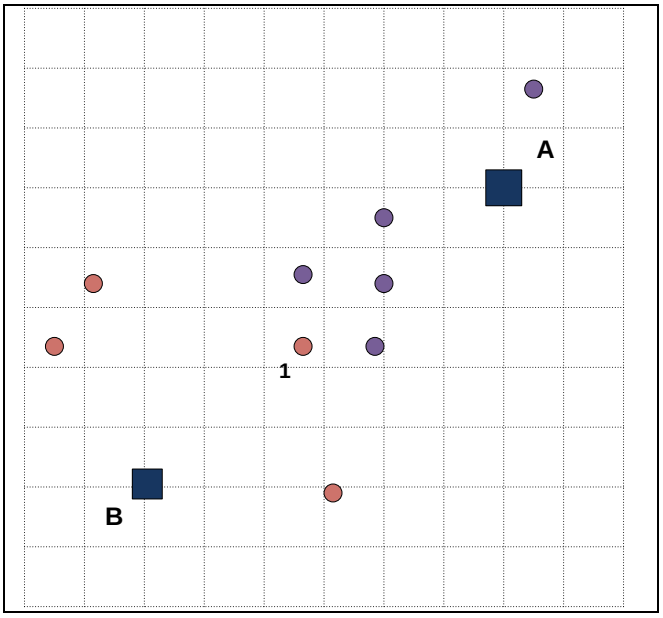

Figure 2: Stop allocation to depot

In the figure above we can see that:

- Stop number 1 is closer to depot B than depot A, and thus allocated to depot B with the simple distance based allocation formula.
- All the neighbouring stops will be serviced by depot A, because of their distance to A.
- The relation of stop 1 with other stops allocated to depot B is much weaker than its relation with the stops from A.

This solution will result in stop 1 being serviced by a vehicle that has to travel an unnecessary additional distance, just to service stop 1. The problem is that the optimization will not be able to do something about it, and the stop might even end as an orphan. The influence of neighbouring stops should be considered.

To comply with these requirements, we implement a clustering algorithm to assist in the allocation logic. This leads to a new problem; which clustering algorithm to use. There are many clustering algorithms because the notion of a 'cluster' cannot be precisely defined. What constitutes a cluster, or a good clustering, has biases because of the application. The domain knowledge is what constitutes the belief that there are subgroups among a bulk of data about objects. Such beliefs mould the structures used to represent those groups.

Clustering generates concepts, provides generalization, data summarization and is an inductive process. Given a data set, any clustering (produced by an algorithm or a human) is a hypothesis to suggest (or explain) groupings in the data.

What discriminates one hypothesis over another given the same data set? This criterion is what we refer to as the mathematical formulation of the inductive principle. It has also received the name clustering criterion. The logic of applying clustering to the multiple depot problem, instigates the advantage of using clustering on other levels of the problem.

We have formulated a sub problem in our problem space, which will assist in solving the complex problem. The next chapter will discuss the implementation of the clustering algorithm for use of classification of stops to improve knowledge of the problem environment.

2.7 Problem Statement

This study will create a solution for the VRP and some variants with the help of evolutionary algorithms implemented in parallel and build on an adaptive object model approach.

The Ant Colony Optimization technique will be the base algorithm used in this study. The aim is to solve 'any' VRP without utilising previous knowledge of the data environment, constraints and size.

The approach utilise adaptive objects to allow flexibility of implementing problem specific constraints, clustering to assist in quicker analysis of data environment and evolutionary algorithms to adapt during run-time.

2.8 Adaptive Objects

A number of forces shape the way in which software evolves. One is a desire to make programs as reusable as possible. Another is to push configuration decisions out into the data. Yet another is to push such decisions out onto the users. Still another is to defer such these decisions until runtime.

Achieving such an implementation is an evolutionary process. First, the problem is solved for a specific instance of a problem. Later, we may broaden the utility by adding options and parameters. After a while, the domain or business objects come to constitute a program of sorts, which can be dynamically constructed and manipulated by users themselves. During this evolutionary process, descriptions of the data, such as maps of the layouts of data objects, and references to methods or code, are needed to permit these heretofore anonymous capabilities to be accessible during runtime.

Our problem or goal is to design an application that has adaptability. As an object-oriented application evolves, the elements of an object-oriented framework emerge. Where raw, undifferentiated, white-box code once was dynamically pluggable black-box components begin to appear. Internal structure, which was once haphazard becomes better differentiated, and more refined. As such a framework evolves, these elements themselves, together with the protocols and interfaces they expose, come to constitute a domain specific language for the framework's target domain.

The following aspects should be considered:

- Efficiency: Highly dynamic systems can be inimical to efficiency. However, efficiency is often a false idol. For instance, the cost of referencing an object in a remote database may be several orders of magnitude more expensive than accessing a local object, and such overhead may overwhelm secondary concerns, such as the cost of assessors vs. direct variable references.
- Complexity: Complex data structures and code are hard to debug and comprehend. Alas, many programmers are better at creating complexity than simplicity.
- Resources: Dynamic strategies can be costly in terms of space, processing time, secondary storage, etc.

- Flexibility: A program should be versatile, and usable in a variety of contexts. This, in turn enhances:
- Reusability: A versatile, flexible application, or, for that matter, a code-level artefact, should be as reusable as possible. The reuse of such code avoids duplicated effort, eases the learning and comprehension burden of new programmers, and makes maintenance easier, since multiple, redundant copies of essentially the same code need not be maintained.
- Adaptability: It is essential that an artefact be flexible enough so as to confront and address changing requirements. We distinguish several "shades" of adaptability.
- Maintainability: It is important that an artefact be maintainable enough to as to confront and address changing requirements. Code that can't be worked on will lapse into stagnation.
- Tailorability: One size does not fit all. Often, an artefact will not fit the needs of a particular user "off the rack", but can be tailored to do so when certain "alterations" can be made.
- Customizability: Just an artefact can be tailored to a particular user or users; it can be customized to adapt it better for a particular task. This may seem at first to be a lot like tailorability, but we find that distinguishing between forces for change than emanate from individual users and those that arise from taking on different tasks useful.

The problem will be solved on the adaptive object model principle. The advantage of this model should be clear, without compromising the efficiency of the algorithm too much. The framework will be clearly defined for the end user to understand. We approach the implementation of this model in the following areas:

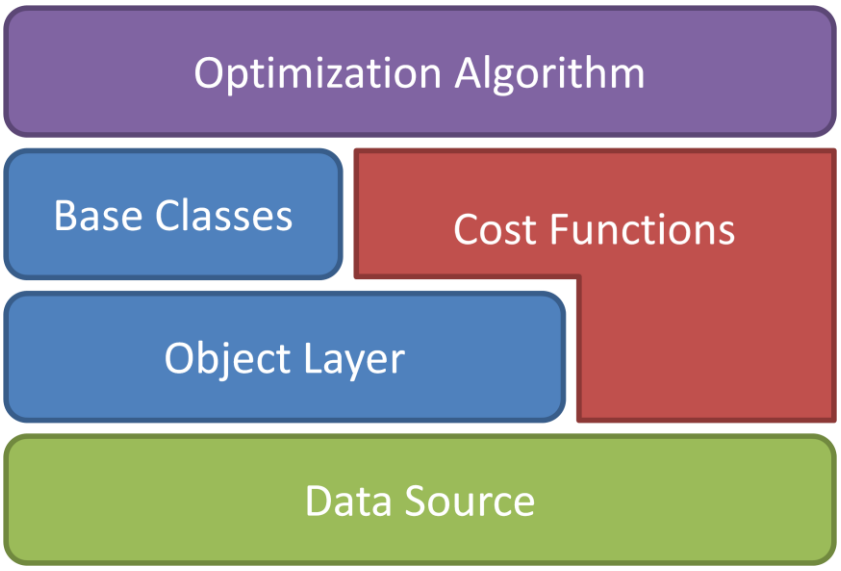

Figure 3: Object Layers

2.8.1 Data source

This layer provides access to the *raw* data and is the responsibility of the implementer. The method of access is not be part of the research and does not influence the solution. It is important that the data is available to load into the objects and can be accessed for any function. The data should also be complete regarding all entities required by the object layer and the cost layer.

The generic component used in this study, the ESI (Expandable Software Infrastructure) utilizes metadata to map the physical data to the data source, which ease the implementation on top of different databases.

2.8.2 Object layer

The object layer represents the source data in a managed and structured way, i.e. the data can be accesses as an object with properties and methods. All the objects are stored in the problem space to be available for usage through the solution. The object layer implements the business entities of the problem instance. A typical problem space in the VRPTW representative problem consist of the following objects and properties, depending on the constraints applied on the problem type:

- Stop – Volume, location, time windows.

- Depot – Time windows, location
- Vehicle – capacity

The object layer can differ from solution to solution and we therefore require a more dependable interface to the algorithm.

2.8.3 Base classes

The input data is loaded into objects and used as the problem space consists of static data with all properties available that is used to solve the problem. The optimization algorithm requires a known interface to work with. From the representative problem formulation, we define the base objects that are required as a list of stops, a depot, a list of vehicles (resources), a list of routes and a solution object. Detail description and explanation follows later in the thesis.

Vehicles do not form part of the original description according to Barbarosoglu and Ozgur (1999), but was identified as a critical enough side constraint input to be part of the base object model. This study re-evaluates the classification of a vehicle as a base class and proposes the classification as an object required to feed a constraint or cost function. The concept formulated stipulates the notion that a route cannot exist without a resource available. This study will utilise a vehicle as the resource, but in an abstraction, a resource can be defined as an entity that must exist before a route can exist.

The problem is to define an interface for the objects that is complete and sufficient enough to be used in the algorithm without limiting the implementer to a fix structure. Any module using this VRP solution that implements the defined interfaces can provide data to the resulting algorithm.

2.8.4 Cost functions

This layer represents the user defined objective cost functions as well as constraint functions. The implementation view a constraint function as checking if a solution is valid, while the cost function is the driver toward a good solution. Combined with a meta-heuristic approach, the cost function is used by the heuristic to determine the quality of the solution and the constraint function to determine the validity of the solution. In the adaptive object model

approach, these two types of functions are the main drivers that support the meta guidance. It is important to remember that these functions are not known beforehand.

2.8.5 Optimization algorithm

The objective function of the problem is to minimize cost while adhering to all constraints. An adaptive object implementation allows for a higher level of abstraction of these functions, i.e. the 'user' of the algorithm has the ability to provide the cost and constraint functions. The design of the algorithm makes use of these external functions to guide the solution to a minimum.

The optimization algorithm has the base classes and cost functions as input. The base class structures are well defined, but the cost function behaviours are unknown for the purpose of this study. The goal is to design an algorithm that can solve the problem with this limited knowledge. The algorithm must build up a knowledge base while executing.

This paper will not discuss the multi-objective cost function, but the resulting solution should be easy to adapt to incorporate such cost drivers. All the underlying layers result in known structures as input for the algorithm.

2.9 Algorithm

In mathematics, computing, linguistics and related subjects, an algorithm is a sequence of finite instructions, often used for calculation and data processing. It is formally a type of effective method in which a list of well-defined instructions for completing a task will, when given an initial state, proceed through a well-defined series of successive states, eventually terminating in an end-state. The transition from one state to the next is not necessarily deterministic; some algorithms, known as probabilistic algorithms, incorporate randomness.

A randomized algorithm or probabilistic algorithm is an algorithm which employs a degree of randomness as part of its logic. In common practice, this means that the machine implementing the algorithm has access to a pseudorandom number generator. The algorithm typically uses the random bits as an auxiliary input to guide its behaviour, in the hope of achieving good performance in the "average case". Formally, the algorithm's performance will be a random variable determined by the random bits, with (hopefully) good expected value;

this expected value is called the expected running time. The "worst case" is typically so unlikely to occur that it can be ignored.

In computer science, a heuristic algorithm or simply a heuristic is an algorithm that ignores whether the solution to the problem can be proven to be correct, but which usually produces a good solution or solves a simpler problem that contains or intersects with the solution of the more complex problem. Heuristics are typically used when there is no known way to find an optimal solution, or when it is desirable to give up finding the optimal solution for an improvement in run time.

Two fundamental goals in computer science are finding algorithms with provably good run times and with provably good or optimal solution quality. A heuristic is an algorithm that abandons one or both of these goals; for example, it usually finds pretty good solutions, but there is no proof the solutions could not get arbitrarily bad; or it usually runs reasonably quickly, but there is no argument that this will always be the case.

Many problems in AI can be solved in theory by intelligently searching through many possible solutions:(Artificial Intelligence, 2008) Reasoning can be reduced to performing a search. For example, logical proof can be viewed as searching for a path that leads from premises to conclusions, where each step is the application of an inference rule (Albus, 2002). Planning algorithms search through trees of goals and sub goals, attempting to find a path to a target goal, a process called means-ends analysis. Robotics algorithms for moving limbs and grasping objects use local searches in configuration space. Many learning algorithms use search algorithms based on optimization.

Simple exhaustive searches are rarely sufficient for most real world problems: the search space (the number of places to search) quickly grows to astronomical numbers. The result is a search that is too slow or never completes. The solution, for many problems, is to use "heuristics" or "rules of thumb" that eliminate choices that are unlikely to lead to the goal (called "pruning the search tree"). Heuristics supply the program with a "best guess" for what path the solution lies on.

A very different kind of search came to prominence in the 1990s, based on the mathematical theory of optimization. For many problems, it is possible to begin the search with some form of a guess and then refine the guess incrementally until no more refinements can be made.

These algorithms can be visualized as blind hill climbing: we begin the search at a random point on the landscape, and then, by jumps or steps, we keep moving our guess uphill, until we reach the top. Other optimization algorithms are simulated annealing, beam search and random optimization

Evolutionary computation uses a form of optimization search. For example, they may begin with a population of organisms (the guesses) and then allow them to mutate and recombine, selecting only the fittest to survive each generation (refining the guesses). Forms of evolutionary computation include swarm intelligence algorithms (such as ant colony or particle swarm optimization) and evolutionary algorithms (such as genetic algorithms and genetic programming).

This thesis will focus on ant colony optimization techniques. The ant colony optimization algorithm (ACO) is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs and is a member of swarm intelligence methods, and it constitutes some meta-heuristic optimizations.

2.9.1 Initial Solution

The general two-phased approach in solving a problem through heuristic methods consists of an initial solution, followed by an improvement stage. High quality initial heuristics often allow local searches and meta-heuristics to achieve better solutions more quickly. The applied solution uses a similar approach and this paragraph formulates the initial solution part of the problem.

Current initial solutions focus on providing quality through minimizing the cost function. Marius Solomon was one of the first researchers to consider the VRPTW. He designed and analyzed a number of algorithms to find initial feasible solutions for the VRPTW (Solomon, 1987). His sequential insertion heuristic (SIH) gave very good results in most environments, and most current heuristic methods make use of this heuristic (or a variation thereof) to effectively find a feasible starting solution.

There exist a number of route construction heuristics. Joubert and Claasen (2006) discuss some of the most prominent improvements on the SIH. All these improvements is related to a specific aspect of the VRP type solved for that instance of the problem, for example, using

the Time Window Compatibility (TWC) depends on the problem to include time windows as a constraint. This research interprets the improvement of the initial solution algorithm as the calculated move of a constraint to this part of the algorithm. Implementing the TWC improvement on the initial solution is born out of the time window constraint.

This research implements an initial solution that utilizes the defined constraints as improvement on the initial solution. The aim is to reduce the number of unnecessary calculations while generating a high quality solution.

The purpose of the initial solution is to provide the improvement stage with a start. We consider the definition of a high quality solution as a solution that is aligned with the improvement stage. This is identified as having a solution that allows quicker convergence. We argue that using the knowledge gained during the initial stage can benefit the improvement stage.

The approach is now to define the outcome of an initial solution as the solution, as well as additional information gained. This interface to the improvement stage is flexible to include scenarios where the user just needs the improvement stage, for example, when a legacy system contains existing routes. The legacy solution can be implemented as the initial solution without any additional information.

2.9.2 Meta-Heuristics

The implementation of an algorithm that can efficiently and in reasonable time solve the aforementioned problem has not been successfully implemented before. To embark on a journey to find a sufficient algorithm requires investigation of existing problems and solutions as well as inventing new methods. Several papers have been presented that solve the VRP with additional side constraints. They mainly focus on solving the basic VRP with one or two additional side constraints. Some of the most popular problems include the VRP with time windows and the VRP with pickup and delivery.

Heuristic methods play an important role in solving problems with this complexity. Most solutions include a heuristic method, or a hybrid of heuristic methods at the heart of the solution. In the next section, we will discuss some of the more popular heuristic methods.

Meta-heuristics, or global optimization heuristics, have a common feature: they guide a subordinate heuristic in accordance with a concept derived from artificial intelligence, biology, nature or physics to improve their performance.

Meta-heuristics succeed in leaving the local optimum by temporarily accepting exchanges that decrease the objective function value. Meta-heuristics use information of the problem environment and the nature of the objective function to direct the search process to regions that promise better solutions. It is possible that the meta-heuristic will return to the local optimum without finding a better solution. This is called cycling and can be avoided by adjusting the heuristic's settings to allow more degrading moves for longer.

The concept of a heuristic being trapped at a local optimum can be demonstrated in Figure 4. If a heuristic finds a solution S, with objective function F(S), where S is close to point C, then it will only improve until it gets to local optimum C. No further improvements in the objective function will be achievable, because all moves will reduce the objective function. However, if a meta-heuristic finds a solution close to point B, degrading moves will be allowed that may direct the search to the global optimum, point A.

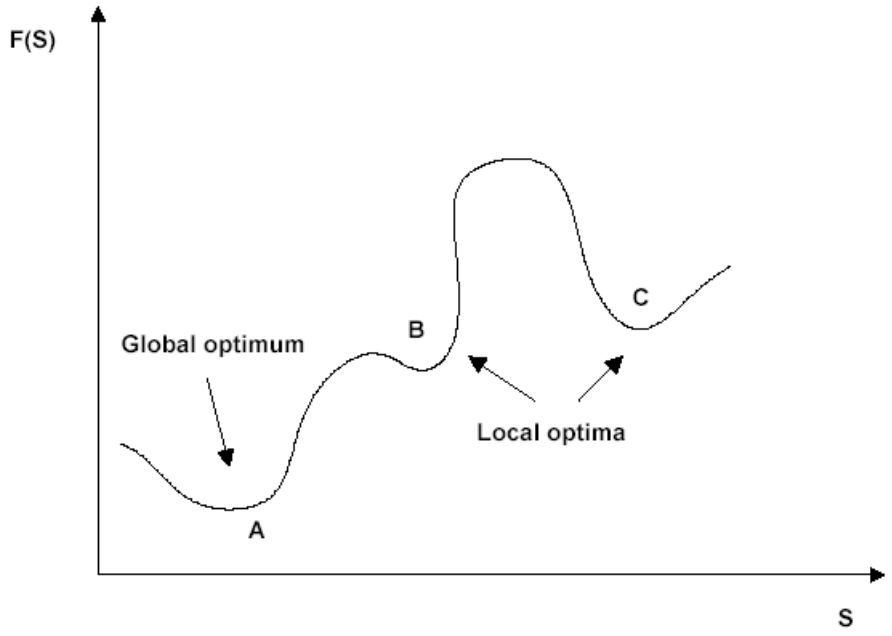

Figure 4: Global and Local Optima

Meta-heuristics will be successful on a given optimization problem if it can provide a balance between the exploitation of the accumulated search experience and the exploration of the search space to identify regions with high quality solutions in a problem specific, near optimal way. The various meta-heuristics are classified according to the following criteria:

- **Trajectory methods vs. discontinuous methods**: Trajectory methods like Simulated Annealing (SA) and Tabu Search (TS) follow one single search trajectory corresponding to a closed walk on the neighbourhood graph. Discontinuous methods allows larger jump in the neighbourhood graph.
- **Populated-based vs. single-point search**: In single-point search only one single solution is manipulated at each iteration of the algorithm. TS and SA are single-point search methods. Genetic and ant colony algorithms are population-based.
- **Memory usage vs. memoryless methods**: Meta-heuristics with memory are the TS, GA and ant systems. According to Taillard et al.(2001) these meta-heuristics with memory can be viewed as adaptive memory programming (AMP) heuristics. The term "memory" was used explicitly for TS, but other meta-heuristics use mechanisms that can be considered as memories. There are meta-heuristics that cannot be entered into the AMP methods, such as SA. However they may be included in the improvement procedure of AMP.
- **One vs. various neighbourhood structures**: SA and TS algorithms are based on one single neighbourhood structure. Other algorithms such as Iterated Local Search typically use at least two different neighbourhood structures.
- **Dynamic vs. static objective function**: Some algorithms modify the evaluation of the single search states during the run of the algorithm. In the use of a dynamic objective function penalties for the inclusion of certain solution attributes that modify the objective function are introduced. TS may be interpreted as using dynamic objective function, as some point in the search is forbidden, corresponding to infinitely high objective function values. The other algorithms use static objective functions.

Evaluation of heuristic methods consists of comparing criteria such as running time, quality of solution, ease of implementation, flexibility and robustness. For the purpose of our algorithm, flexibility is an important consideration. The algorithm should be able to handle

changes in the data patterns, side constraints and objective function, as each client has his own specific requirement. We are not working on a predetermined set of data with a specified objective function. Working in such an environment makes it possible to find a method that is effective for that specific environment by making use of the knowledge about the problem.

Because the heuristic methods are non deterministic, i.e. we cannot predict the result even if we apply the same algorithm on the same data with the same number of iterations, the algorithm should not perform poorly on any instance, as well as being able to produce a good solution each time it is applied to the same instance.

We will also try to validate the applicability of the method on our problem by discussing the design of the method is well as what we see as its advantages and disadvantages. With this approach we will filter out certain methods. Comparisons discussed in this paper are from existing papers, which mainly present the best results found for the method. Comparison is also made difficult because solutions were not all implemented on the same computer (running time), and have not all use the same number of iterations. Existing methods is also not designed for our specific problem and thus we cannot really compare methods outright to decide on a method to implement for our problem.

Using only the best results of a non-deterministic heuristic, as is often done in the literature, may create a false picture of its real performance. We considered average results based on multiple executions on each problem an important basis for the comparison of non-deterministic methods. Furthermore, it would also be important to report the worst-case performance.

Moreover, an algorithm should be able to produce good solutions every time it is applied to a given instance. This is to be highlighted since any heuristics are non-deterministic, and contain some random components such as randomly chosen parameter values. The output of separate executions of these non-deterministic methods on the same problem is in practice never the same. This makes it difficult to analyze and compare results.

The meta-heuristic method has a guidance procedure of some sort to help it traversing through the solution space. A meta-heuristic is the implementation of a heuristic method with a guidance procedure.

2.9.3 Tabu Search

Tabu Search was proposed by Glover (1986) and has quickly become one of the best and most widespread local search methods for combinatorial optimization. The method performs an exploration of the solution space by moving from a solution x_t identified at iteration t to the best solution x_{t+1} in a subset of the neighbourhood $N(x_t)$ of x_t. Since x_{t+1} does not necessarily improve upon x_t, a tabu mechanism is put in place to prevent the process from cycling over a sequence of solutions.

A naïve way to prevent cycles is to forbid the process from going back to previously encountered solutions, but doing so would typically require excessive bookkeeping. Instead, some attributes of past solutions are registered and any solution possessing these attributes may not be considered for μ iterations. This mechanism is often referred to as short term memory. Other features such as diversification and intensification are often implemented. The purpose of diversification is to ensure that the search process will not be restricted to a limited portion of the solution space. It keeps track of past solutions and penalizes frequently performed moves. This is often called long term memory. Intensification consists of performing an accentuated search around the best known solutions.

There are two main ways to define neighbourhood structures in Tabu Search algorithms for the VRP. The first, termed λ-interchanges by Osman (Osman, 1993) consist of exchanging up to λ customers between two routes. The second, called ejection chains typically acts on more than two routes at the same time.

To prevent cycling, it is common to prohibit reverse moves for μ iterations. Thus a customer moved from route r to route s at iteration t may be prohibited from being reinserted in route r until iteration $t + \mu$, or it may not leave route s until iteration $t + \mu$. Taillard (1999) suggests randomly selecting μ in an interval $[\mu; \mu]$, according to a discrete uniform distribution, but some algorithms use a fixed value of μ. It is also common to use an aspiration criterion to revoke the tabu status of a move if this causes no risk of cycling, for example if this yields a better overall incumbent feasible solution, or a better incumbent among the set of solutions possessing a certain attribute.

To diversify the search, most Tabu Search implementations penalize frequently performed moves. This is done by adding to the routing cost $c(x_{t+1})$ of x_{t+1} a penalty term equal to the product of three factors:

1. a factor measuring the past frequency of the move;
2. a factor measuring instance size (such as $\sqrt{n}$);
3. a user-controlled scaling factor.

In practice, implementing such a mechanism is both effective and computationally inexpensive.

Intensification consists of accentuating the search in promising regions. Periodic route improvements by means of TSP algorithms may be classified as intensification techniques. Another way of performing intensification is to conduct a search using a wider neighbourhood structure around some of the best known solutions.

An adaptive memory is a population of good solutions encountered during the search procedure. Similar to what is done in genetic algorithms; the idea is to combine solution elements from the population to construct new solutions. If their value is less than those of some solutions in the population, they are retained and the worst solutions are discarded from the population.

In the VRP context, a new solution is initialized from a number of non-overlapping routes extracted from good solutions. Typically these routes do not cover all vertices for otherwise there would be overlaps. The search is then initiated from the selected routes and the un-routed vertices.

2.9.4 Ant Algorithms.

Ants appeared on earth some 100 million years ago, and have a current total population estimated at 10^{16} individuals. Most of these ants are social insects, living in colonies of 30 to millions of individuals. The complex behaviours that emerge from colonies of ants have intrigued humans, and there have been many studies of ant colonies aimed at a better understanding of these collective behaviours. Collective ant behaviours that have been

studied include the foraging behaviour, division of labour cemetery organization and brood care, and construction of nests. (Engelbrecht, 2007)

The South African, Eugene Marais (1871-1936) was one of the first to study termite colonies. In 1910, Marais abandoned his law practice and retreated to the remote Waterberge – the mountain area north-west of Pretoria. Here he studied two creatures - termites and baboons which, on the face of it, had nothing in common. Both fascinated him, as did all wild creatures. (Marais, n.d.)

He published his conclusions about termites as a series of speculative articles, written entirely in Afrikaans and appearing only in local newspapers, as The Soul of the White Ant. While observing the natural behaviour of these creatures, he noticed that firstly, the whole termitarium had to be considered as a single organism whose organs work like those of a human being. The queen was the brain and the womb; the workers were mouthparts and tissue builders; the soldiers' white blood cells and the humus gardens were the stomach. And secondly that the actions within the termitarium were completely instinctive.

Marais began writing Soul of the Ape in 1916, but never finished it. It was published posthumously years later. His theory was that, unlike termites, baboons – and by extension all primates – had the ability to memorize the relationship between cause and effect. They could therefore vary their behaviour voluntarily. While termites were instinctive, the mind of baboons was based on "causal memory".

The reason for this difference, according to Marais, was natural selection. According to him, natural selection was not, like Darwin had insisted, the survival of the fittest, but rather the line of least resistance. Those species best able to adapt to their specific environment survived, while those not able to, would become extinct. Natural selection, therefore, had the tendency to both localize and specialize species.

These conclusions to which he came were new and radical and might well have had an influence in Europe. But Marais was half a hemisphere away, half a century too soon and writing in a language no one could understand. The Soul of the White Ant was brought under the attention of the world only by being seemingly plagiarized by a Belgian Nobel prize winner, Maurice Maeterlinck. The Soul of the Ape was incomplete and originally only published in South Africa.

While most research effort concentrated on developing algorithmic models of foraging behaviour, models have been developed for other behaviours, including division of labour, cooperative support, self-assembly, cemetery organization and even traffic flow. These complex behaviours emerge from the collective behaviour of much unsophisticated individuals.

In the context of collective behaviour, social insects are basically stimulus-response agents. Based on information perceived from the local environment, an individual performs a simple, basic action. These simple actions appear to have a large random component. Despite this simplicity in individual behaviour, social insects form a highly structured social organism.

One of the most surprising behavioural patterns exhibited by ants is the ability of certain ant species to find the shortest path (Dorigo and Stutzle, 2004). Biologists have shown experimentally that this is possible by exploiting communication based only on pheromones. It is these behavioural patterns that inspire the development of optimization algorithms based on the ant behaviour. The first attempts in this direction appeared in the early '90s and can be considered as rather "toy" demonstrations. Since then these and similar ideas have attracted more research which let to Ant Colony Optimization (ACO) algorithms.

How do ants find the shortest path between their nest and food source, without any visible, central, active coordination mechanisms? Studies of the foraging behaviour of several species of real ants revealed an initial random or chaotic activity pattern in the search for food. As soon as a food source is located, activity patterns become more organized with more and more ants following the same path to the food source. "Auto-magically", soon all ants follow the same shortest path.

This emergent behaviour is a result of a recruitment mechanism whereby ants that have located a food source influence other ants towards the food source. The recruitment mechanism differs for different species, and can either be in the form of direct contact, or indirect "communication". Most ant species use the latter form of recruitment, where communication is via pheromone trails. When an ant locates a food source, is carries a food item to the nest and lays pheromone along the trail.

Forager ants decide which path to follow based on the pheromone concentrations on the different paths. Paths with a lager pheromone concentration have a higher probability of

being selected. As more ants follow a specific trail, the desirability of that path is reinforced by more pheromone being deposited by the foragers, which attracts more ants to follow that path. The collective behaviour that results is a form of autocatalytic behaviour, where positive feedback about a food path causes that path to be followed by more and more ants. The indirect communication where ants modify their environment (by laying of pheromones) to influence the behaviour of other ants is referred to as stigmergy.

Deneubourg et al (1990) studied the foraging behaviour of the Argentine ant species *Iridomyrmex humilis* in order to develop a formal model to describe its behaviour. In the double bridge experiment, a nest of a colony of Argentine ants is connected to a food source by two bridges. The ants can reach the food source and get back to the nest using any of the two bridges. The goal of the experiment is to observe the resulting behaviour of the colony. What is observed is that if the two bridges have the same length, the ants tend to converge towards the use of one of the two bridges.

If the experiment is repeated a number of times, it is observed that each of the two bridges is used in about 50% of the cases. These results can be explained by the fact that, while moving, ants deposit pheromone on the ground; and whenever they must choose which path to follow, their choice is biased by pheromone: the higher the pheromone concentration found on a particular path, the higher is the probability to follow that path.

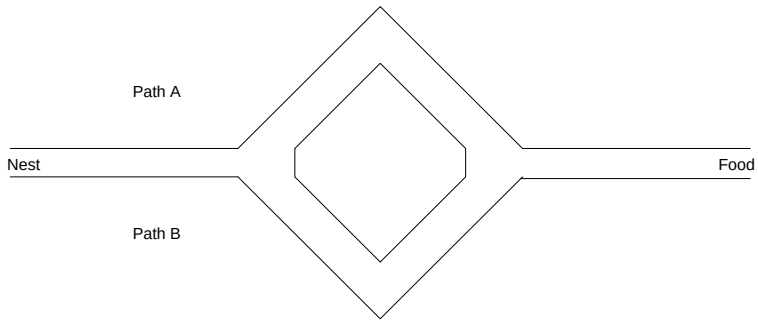

Figure 5: Binary Bridge Experiment

From this experiment, (illustrated in Figure 5), a simple formal model was developed to characterize the path section process. For this purpose, it is assumed that ants deposit the same amount of pheromone and that pheromone does not evaporate. Let $n_A(t)$ and $n_B(t)$

denote the number of ants on paths A and B respectively at time step *t*. Pasteels, Deneubourg and Goss (1987) found empirically that the probability of the next ant to choose path *A* at time step *t +1* is given as,

$$P_A(t+1) = \frac{(c + n_A(t))^\alpha}{(c + n_A(t))^\alpha + (c + n_B)^\alpha} = 1 - P_B(t+1)$$

Equation 1: Binary Bridge Probability Equation

Where *c* quantifies the degree of attraction of an unexplored branch, and α is the bias to using pheromone deposits in the decision process. The larger the value of α, the higher the probability that the next ant will follow the path with a higher pheromone concentration – even if that branch has only slightly more pheromone deposits. The larger the value of *c*, the more pheromone deposits is required to make the choice of path non-random. It was found empirically that $\alpha \approx 2$ and $c \approx 20$ provides a best fit to the experimentally observed behavior.

Using the probability defined in Equation 1, the decision rule of an ant that arrives at the binary bridge is expressed as follows: $if\ U(0,1) \leq P_A(t+1)\ then$ follow path A otherwise follow path B.

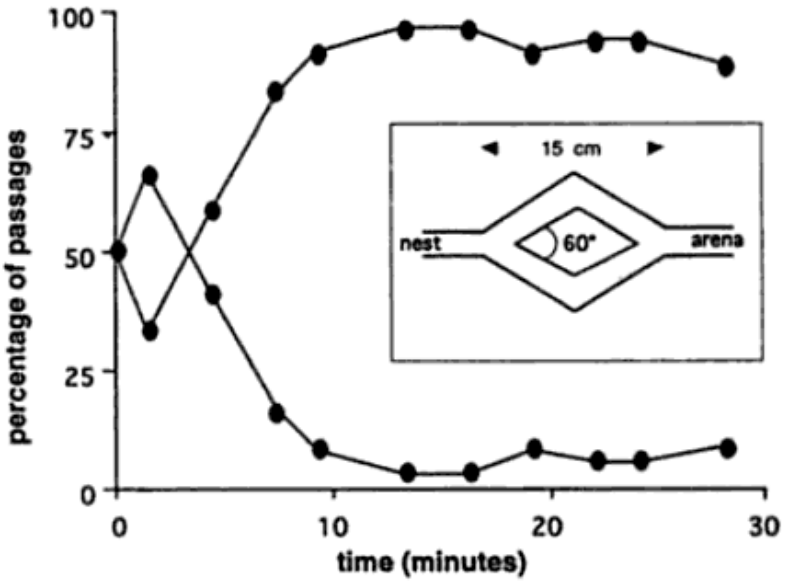

Figure 6: Percentage of all passages per unit time as a function of time.

The winning branch in time. Figure was not favoured by the initial fluctuations, which indicates that these fluctuations were not strong enough to promote exploitation of the other branch in the beginning (Bonabeau, Dorigo and Theraulaz, 1999) .

Goss et al (1989) extended the binary bridge experiment, where one of the branches of the bridge was longer than the other, as illustrated in Figure 7. Dots in this figure indicate ants. Initially, paths are chosen randomly with approximately the same number of ants following both paths (as illustrated in Figure 7 on the left). Over time, more and more ants follow the shorter path as illustrated in Figure 7 on the right. Selection is biased towards the shortest path, since ants that follow the shortest path returns to the earlier than ants on the longer path. The pheromone on the shorter path is therefore reinforced sooner than that on the longer path.

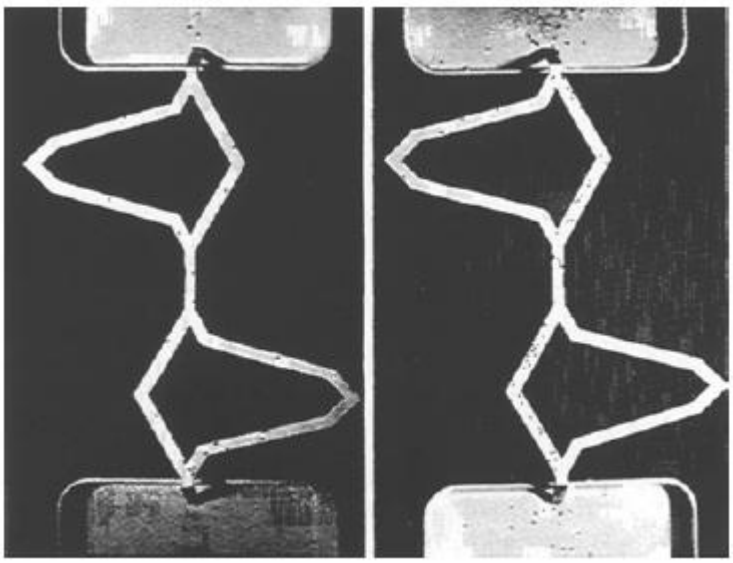

Figure 7: Shortest Path Selection by Forager Ants

Goss et al found that the probability of selecting the shorter path increases with the length ratio between the two paths.

Although an ant colony exhibits complex adaptive behaviour, a single ant follows very simple behaviours. An ant can be seen as a stimulus-response agent (Nilsson, 1998): the ant observes pheromone concentrations and produces an action based on the pheromone-stimulus. An ant can therefore abstractly be considered as a simple computational agent. An artificial ant algorithmically models this simple behaviour of real ants. The logic implemented is a simple production system with a set of production rules as illustrated in Algorithm 1. This algorithm is executed at each point where the ant needs to make a decision.

```
Let r ~ U(0,1);
For each potential path A do
      Calculate P_A using e.g. Equation 1;
      If r ≤ P_A then
            Follow path A;
            Break;
      End
End
```

Algorithm 1: Artificial Ant Decision Process

While Algorithm 1 implements a simple random selection mechanism, any other probabilistic selection mechanism can be used, for example, roulette wheel selection. The implementation of an Ant solution on the VRP necessitate a more complex method as the cost function that drives the probability depends on more combinations.

These experimental observations form a basic set of tasks any model needs to explore. The following results were obtained using a few different species of ants in the bridge experiment.

1. When ants are exposed to two paths of unequal length the ants will choose the shortest path.
2. If a shorter path is offered after the ants have chosen, they are unable to switch to the new path.
3. The ants will break symmetry and chose one path, even when both paths are equal.
4. If ants are offered two unequal food sources they will usually choose the richest source.
5. If a richer food source is offered after the ants have chosen, some species can switch to this new source, and others are unable to.

In line with the ideas proposed above, the most important modelling constraint in what follows is the principle of locality. In the models we will use, the behaviour of the individual organisms will be determined solely by local influences. This means that the individual organisms will not have any memory, non-local navigational skills, or any type of behaviour that involves storage of internal information. Any information flow must then be a product of the collective behaviour.

2.9.5 Memetic Algorithms

Memetic Algorithms (MA) are used as a synergy of evolutionary or any population-based approach with separate individual learning or local improvement procedures for problem search. Cultural evolution, including the evolution of knowledge, can be modelled through the same basic principles of variation and selection that underlie biological evolution. This implies a shift from genes as units of biological information to a new type of units of cultural information: memes.

A meme is a cognitive or behavioural pattern that can be transmitted from one individual to another one. Since the individual who transmitted the meme will continue to carry it, the transmission can be interpreted as a replication: a copy of the meme is made in the memory of another individual, making him or her into a carrier of the meme. (Dawkins, 1976)

Memetic Algorithms denote a family of meta-heuristics that have as central theme the hybridization of different algorithmic approaches for a given problem. Special emphasis was given to the use of a population-based approach in which a set of cooperating and competing agents were engaged in periods of individual improvement of the solutions while they sporadically interact. Another main theme was to introduce problem and instance-dependent knowledge as a way of speeding-up the search process. MAs exploit problem-knowledge by incorporating pre-existing heuristics, pre-processing data reduction rules, approximation and fixed-parameter tractable algorithms, local search techniques, specialized recombination operators, truncated exact methods, etc. Also, an important factor is the use of adequate representations of the problem being tackled. (Moscato and Cotta, 2005)

2.10 Parallel

Searching the solution space for the best solution can be improved through a parallel implementation. The algorithm goal is to increase the diversification of the probable solutions as well as increase efficiency. Local searches tend to get stuck in a local minimum. With a parallel method, this negative aspect can be utilized. We would like the solution to reach a local minimum as soon as possible, in as many as possible places.

The evaluation of different solutions provides some new challenges. The goal is to provide solutions that are diverse in the solution space. This requires a method to compare the diversity measure of solutions. If successfully applied, the method accomplishes a dual goal: assisting with guiding the algorithm into other areas to explore, and providing the end result with alternative solutions for the problem.

The diversification comparison assist the parallel algorithm not to pursue venues simultaneously that will ultimately end up in the same solution. It also provides the meta control mechanism powerful information regarding the tightness of the problem. If there exist a number of solutions that qualify as different from each other, it indicates that the solution space contains multiple possibilities of valid solutions. If not, the problem space is

tight and more emphasis should be put on the local search which finds the most optimal solution for that particular stream.

Providing the user with a diverse set of solutions that can be viewed as close to each other in the total cost, allows the user to execute a selection based on non-tangible variables that was not considered in the algorithm. These variables can range from a biting dog to a possible new customer to join a route. It can also assist in strategic planning by highlighting master routes through execution of different scenarios.

This thesis aim to design a parallel model for algorithm to solve various instances of the VRP on various data environment scenarios. This parallel design is an extension of the proposed adaptive algorithm. The focus is mainly on performance and efficiency, but discussion will include possible future implementation. The parallel solution assists in generating a number of diverse feasible solutions, which provide a choice for the user.

The programming style used is a synchronous master/workers paradigm on multiple levels. The master implements a central memory through which passes selective communication to share between the diverse solutions, and that captures the global knowledge acquired during the search. The worker implements the search process. The parallel algorithm continues from the initial setup which creates more than one initial solution through various methods.

The pheromone matrix and the best found solutions will be managed by the master. At specified iterations, the master consolidates the pheromone matrix from all the workers and then broadcast the pheromone matrix to all the workers. Each worker handles an ant process. He receives the pheromone matrix, constructs a complete solution, applies a tabu search for this solution and sends the solution found to the master. When the master receives all the solutions, he updates the pheromone matrix and the best solutions found, and then the process is iterated.

2.11 Summary

This thesis identifies the problem of solving the VRP with variants through applying an adaptive solution which has no predefined knowledge of the problem environment. The environment is defined as the geographical distribution of the stops, the constraints applied on the solution and the objective cost functions to minimize.

The current focus on solving VRP is to identify the type of problem beforehand and implement as solution for the specific type. Better computer processing power allows the introduction of alternative methods into the solution. This research utilizes concepts such as clustering, adaptive object modelling, meta-heuristics and adaptive object modelling to improve the quality of the solution. Parallel implementation is considered throughout to improve speed.

Success is measured if an answer is *good enough* in a *reasonable time* for a problem where the user define the *constraint* and *cost* model.

3 ADAPTIVE OBJECT MODELING

3.1 Overview

The study formulates a new angle on an already challenging problem to solve. It exploits methods available to approach the problem as well as incorporating traditionally used methods. This chapter provides the detail of the design for the adaptive object model. It introduces mechanisms using adaptive objects and data structures to achieve the composition of the problem formulated.

This part of the thesis is important for the deployment of the problem that can benefit the industry. The design of an Information Technology System should focus on the following key performance indicators (Schulman, August 2002):

- Efficiency - *accomplishment of* or *ability to accomplish* a job with a minimum expenditure of time and effort
- Effectiveness - adequate to accomplish a purpose; producing the intended or expected result
- Alignment - a state of agreement or cooperation among persons, groups, nations, etc., with a common cause or viewpoint
- Agility - the power of moving quickly and easily
- Integration - an act or instance of combining into an integral whole.

Current study on the VRP focuses mostly on the first two indicators because it does not consider the problem as part of an enterprise system. The aim of this study requires us to view the reverse order of these indicators.

Integration, both within and among cooperating enterprises, now comes first and is most important, providing the highest value. Dynamic integration creates the ability for many enterprises to participate within IT solution. Agility, the ability to react quickly, comes second. Alignment contribute to the linking of IT and business goals. Effectiveness and efficiency are important, but "good enough" gets us to where we need to go now, and getting to business goals quickly is better than perfection.

Also, although creating solutions that are durable is important, it is more important to put agility and "evergreening" features into architectures that make them easy to change as new business requirements come along. Creating a design for business change will end up being more important than creating the perfect solution. So, although we are not jettisoning effectiveness and efficiency as KPIs, we are increasing the others' priority and putting them a bit "on the back burner" and shaking up our viewpoint.

To achieve this goal on the micro level, we implement an adaptive object approach, which separate the problem environment. This includes the objective function and constraints from the actual optimization algorithm. It requires a clear communication structure between the discrete components. These components must be well-defined regarding functions, properties and results.

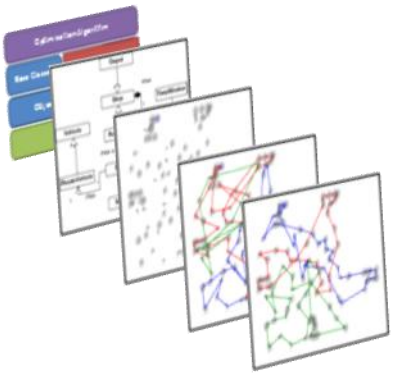

Figure 8: From modelling to solution

Existing solution approaches rely heavily on knowledge of the problem environment to improve efficiency of the algorithm. This research cannot exploit the lessons learned from previous research on problem types as the objective is to solve a problem for any environment. Joubert (2006) apply a method of evaluation first and then selecting an appropriate heuristic. This research is a step in the right direction. The shortfall is in the provision of all types of environments. The approach followed by Joubert relies on the knowledge of a specific problem type, and evaluates only the geographical and time window compatibility.

We will take the first steps towards and adaptive object model solution approach. The proposed method uncovers a new area of defining the VRP. Previous solutions were done on an object-oriented basis (Moolman, 2004). Adaptive software is an extension of object-

oriented software where relationships between functions and data are left flexible, that is, where functions and data are loosely coupled through navigation specifications. Adaptive means that the software heuristically changes itself to handle an interesting class of requirements changes related to changing the object structure.

Adaptive software is a natural evolution of object-oriented software since every object oriented program is essentially an adaptive program. In many cases however, the adaptiveness of the object-oriented program can be significantly improved. Although object-oriented programs are easier to reuse than programs that are not written in an object-oriented style, object-oriented programs are still very rigid and hard to evolve. Our experience shows that for most application domains, object-oriented programs can be made significantly more general and extensible by expressing them as adaptive programs. An adaptive program allows us to express the ***intention*** of a program without being side-tracked by the details of the object structure (Lieberherr, 1996).

Object-oriented programming is a promising technology that has been developed over the last twenty years. One important advantage of object-oriented programming is that it reduces the **semantic gap** between a program and the world it models because the world consists of physical and abstract objects that are represented naturally by software objects in an object-oriented program. However, object-oriented design and programming has several disadvantages, the most significant of which is that it binds functions and data too tightly.

A loose binding between functions and data allows very generic software where data structure information in the functions or procedures is only used to constrain the applicable data structures. Before a program can be run, we select one of the applicable data structures, which in turn usually determine the structure of the input objects. The goal when writing the functions is to minimize the assumptions we make about the data structures. This technique could be called data-structure-shy programming, and it leads to generic software that can be flexibly customized later. One data-structure-shy program potentially describes an infinite collection of object-oriented programs.

In the following paragraphs the domain data is discussed from a generic point of view. The structure of the data is fixed on the domain level.

3.2 Supply chain domain

The VRP form part of a supply chain and should be integrated into the planning solutions for a supply chain. Supply chain management is among the most complex and difficult activities in today's environment of shorter lead times, tighter delivery schedules, and dramatically increased product variety. It is also among the most important. We view the usage of the logistics optimization on three levels:

- Strategic – This level represents the plan of action intended to accomplish a specific goal. Strategic Supply Chain Management explores the knowledge, techniques, and strategies necessary to create value and achieve competitive advantage from your supply chain.
- Tactical – This level represents a manoeuvre for achieving a goal. Tactical supply chain decisions focus on adopting measures that will produce cost benefits for a company. Tactical decisions are made within the constraints of the overarching strategic supply chain decisions made by company management. It contains more definite information for scenarios.
- Operational – This level implements a series of actions for achieving a result. Operational supply chain decisions are made hundreds of times each day in a company. These are the decisions that are made at business locations that affect how products are developed, sold, moved and manufactured. Operational decisions are made with awareness of the strategic and tactical decisions that have been adopted within a company. The day to day operational supply chain decisions ensure that the products efficiently move along the supply chain achieving the maximum cost benefit. A number of examples of operational decisions can be identified in manufacturing, supplier relationships and **logistics.**

This study acknowledges the usage of the VRP on all the levels. On the higher level implementation, input parameters are manipulated by scenario generating tools. This chapter will clearly specify the interfaces to the outside supply chain processes to allow for seamless integration and maximum use.

It is important to understand the link to the supply chain objects. The adaptive object model will feed from a base domain environment. For the VRP, we define that environment as the supply chain logistics. The supply chain objects are described as follows:

Entity	Description
Commodity	Commodity is a product or service for which there exists demand. In the supply chain logistics context, we are interested in specific properties such as weight, volume and certain constraints pertaining to the distribution of the commodity. Solving the VRP for multiple commodities can be handled in the proposed framework if the commodities use the same unit of measurement, e.g. weight or volume or when they are mapped as a multi-objective function. This study does not implement a multi-objective scenario. Additional requirements can be enforced through constraints functions, e.g. fresh and frozen product that is not allowed to share a resource.
Contract	A contract is an agreement between two or more parties for the doing or not doing of something specified. Contracts provide clients which have locations and are used for the routing. Contracts assist with the management of when a client is not valid to serve anymore.
Contract Line	The contract line represents the detail regarding the contract on a specific commodity. A contract can contain more than one commodity. A contract line indicates the activity required, e.g. supply, demand, returns, etc. The input into the algorithm considers only one activity type per solution. The contract line specifies the location of the activity required. The location should relate to a distribution centre either through manual allocation or through a pre-process algorithm before the algorithm start. The algorithm will handle only one distribution centre as input on the lowest level. The contract line contains a link to quantities required per occurrence. An occurrence is the date when the activity must be

	done for the contract line and is calculated from a schedule. The occurrence contains the time windows for the service of the location.
Schedule	A schedule represents the date for which a contract line is active as well as the frequency of visits for the specific contract line, e.g. for the period March to June service the location only Mondays and Fridays.
Units of measurement	A unit of measurement is a standardized quantity of a physical property, used as a factor to express occurring quantities of that property. The scenarios in this research assume that the defined comparison constraints do not require a unit conversion. The commodity demand is specified in the unit of measurement.
Locations	In geography, location is a position or point in physical space that something occupies on Earths' surface. A real location can often be designated using a specific pairing of latitude and longitude, a Cartesian coordinate grid (e.g., State Plane Coordinate System), a spherical coordinate system, or an ellipsoid-based system (e.g., World Geodetic System). A location may be described as either absolute location, meaning the exact location of an object, or relative location, meaning the location of one object relative to another and another or in a general area. There are two types of location, relative & absolute. Relative deals with the relative spot of something on Earth. Absolute deals with the exact spot of something on Earth. Locations has played an important role in interpreting a VRP problem, but must be seen as source data for visual presentation and possible input to cost or constraint functions.
Resource Types	A resource is any physical or virtual entity of limited availability, or anything used to help one earn a living (solve the problem). It

	is important to note the possibility of different resource types for future studies. This thesis will consider only one resource type. Vehicles are the typical resource type for the VRP.

Table 2: Supply Chain Entities

The supply chain problem spans over a period of time and can depend on forecasts, seasonal trends, weather patterns, etc. The proposed VRP solution is for a single instance of routing to be done from a depot. The implementation of the solution requires the domain expert to prepare the data for the solution.

3.3 Components

The previous paragraph described the typical domain environment on top of which the solution will be implemented. We can now revisit the component model as explained in chapter 2. This paragraph discusses the objective of the components and the required interfaces on a high level. It indicates the level of adaptiveness and object-oriented design of each component.

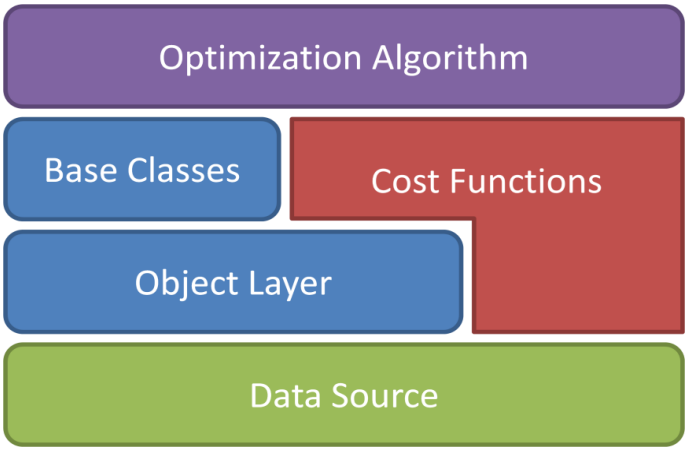

Figure 9: Solution component breakdown

Adaptiveness is achieved by expressing programs as loosely coupled, cooperating fragments, and each one describing only the concerns of one context. The loose coupling is achieved with novel means such as concise object navigation specifications. The loose coupling of the fragments leads to adaptiveness since many changes in one fragment preserve the intent of the other cooperating fragments, which then adjust automatically to the changed fragment.

Adaptive software works with partial knowledge about class structures that directly supports an iterative software life-cycle. Figure 9 displays the different loosely coupled components that implements a combination of object oriented structure and adaptive modelling. The components depict the basic building block hierarchy and dependency.

This study will create some of the components as new methods for solving any VRP related problem. It will provide a guide of how to create other components to ensure that the solution is still effective in time to solve. And it will indicate which components are not in the control of the implementer. The components' services can be described as follows:

Service	Description
Data Source	The physical storage of the data can reside in different formats. The data source provides the interface to the required information.
Object Layer	The object layer interprets the data from the data source into the required domain objects. It applies filters on the data which result in the subset of data required for the problem instance.
Base Classes	The base classes represent the data objects and additional structures used by the algorithm. These objects are data light because of their access to the domain specific object layer.
Cost Functions	The cost and constraint functions reside in objects that are accessible by the algorithm and have access to all underlying data models. These objects encapsulate the complexity of the calculations required. They expose a limited set of function calls that is aligned with the domain model.

Optimisation Algorithm	The algorithm utilise the base classes and cost functions to optimise an unknown VRP type problem.
Process Monitoring and Measurement	All processes are monitored to ensure efficiency of the system and allow for future areas of enhancements.

Table 3: Component blocks

Adhering to the principle of service-oriented architecture, all components are loosely coupled, using well-defined interface protocols, preferably message-oriented. This criterion has the following desirable design qualities which contribute to a lower cost to the implementer.

- Each component can be designed, implemented and tested independently of the other systems based on the agreed interface protocols. Knowledge of the internal design of other systems, such as implementation language, choice of RDBMS etc. is not required.
- Support for effective end-to-end testing.
- Any system that supports the interface protocols may be updated or replaced.
- High cohesion - The structure of the components that form part of the overall architecture is focused on well-defined areas. The responsibility split between components is clear and logical, and it is always clear which component is functioning. This design criterion reduces system complexity.
- Strong encapsulation - Encapsulation hides the details of a system but offers a well-defined interface for exchanging information. Like high cohesion, this design criterion reduces overall complexity but also allows changes to the internal design without impacting on the other systems. Encapsulation also shields the internal data from external tampering.
- Determinism ensures that given a defined component state and a determined sequence of events, the end result is always the same, and thus able to be reproduced in a test environment. This allows effective end-to-end testing and contributes to a reliable system.

The system will be implemented according to the following logical architecture. The diagram shows the interface relations of the building blocks ranging from the raw data to the algorithm.

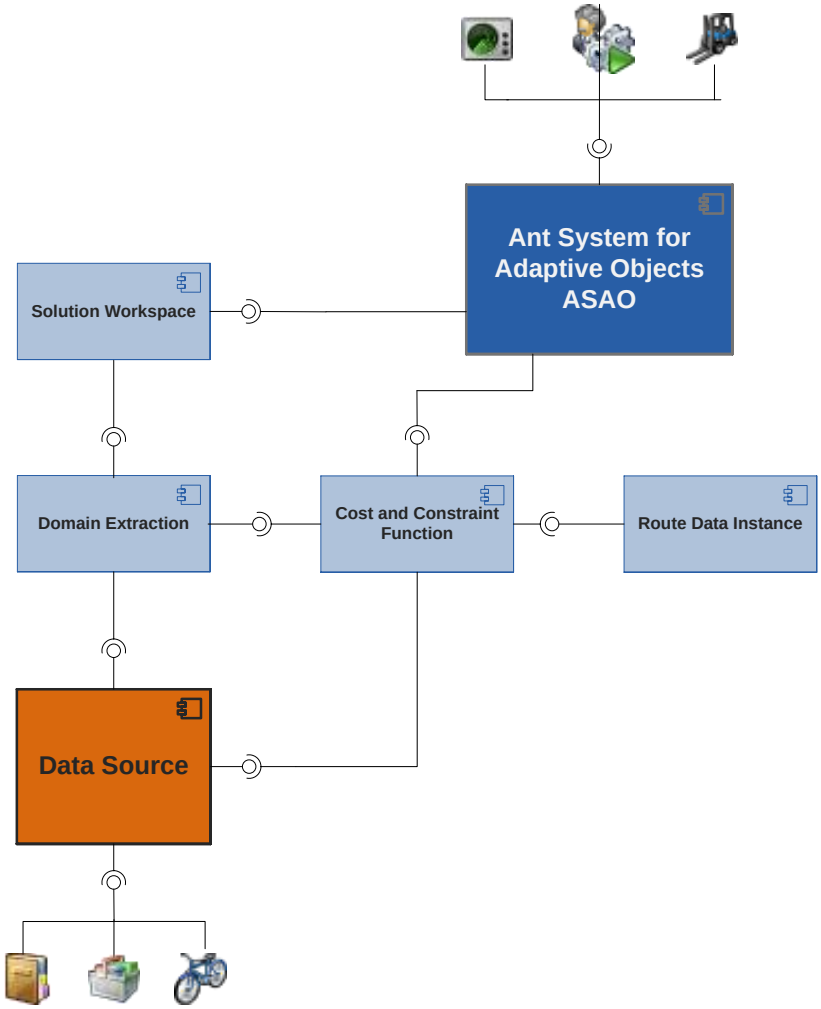

Figure 10: System logical overview

3.3.1 Data source

This component encapsulates the physical data and is provided by the client. A typical example consists of a database that stores all the required data. A more complex scenario is where different information is stored in different places, e.g. forecast data is used to determine the volumes and is stored in the financial system, while customer information is stored in the CRM and is used for location of clients.

The solution is built on the Expandable Software Infrastructure (ESI) framework, which implements an adaptive data model through meta-data. The ESI maps the functional attributes to the physical fields and takes care of the underlying physical structure.

The data source layer provides an abstraction of the physical data for the object layer and the cost functions. The domain implementation on the data source guarantees a base model, but is extendible on attributes. These additional attributes is not known at the design time of the algorithm. This situation is an important factor in designing the optimization algorithm.

The data source component implements the interfaces from the external data providers and presents an interface that is used by the domain objects, as well as the cost and constraint function component.

3.3.2 Domain Objects

The domain object component represents the source data in a managed, structured and accessible way, i.e. the data can be accessed in code as an object with properties, methods and relations to other objects. The identified objects forms the base of our problem space and are accordingly stored in a known problem space environment. The objects are designed specifically for the VRP environment. Further research can extent the objects to fit into a general optimization structure. There exists only one problem space in our solution.

The main purpose of the problem space is to be a placeholder for all the original data and to implement the raw data from the data source into a conceptual structured way that models the ***intent*** of the problem, i.e. the VRP.

An additional benefit of the object layer as placeholder is to provide better memory management, because most of the calculated result classes can now only reference to an entry in the problem space. The design allow for the problem space to be utilize in any kind of solution, e.g. the same problem space can be used to calculate activity based costing, or cluster stops as we've used as an environment evaluating step. This step achieves our goal to keep the design flexible and extendible.

The second purpose of implementing a conceptual model is important in the use of objects in further steps. As described a previous paragraph, the data source is build on a framework that allows for meta-data driven classes. The level of abstraction allows the user to implement

anything in the detail of the base structure. The framework also allow for the definition of a domain model. It is that domain model that comes into play during the object layer implementation.

The first step in defining a domain model is to define what is required in our implementation of the object layer. This requirement is set by the base classes defined in detail later in this chapter. Chapter 2 describes the complete VRP problem. According to Barbarosoglu and Ozgur (1999) the VRP can be described as the problem of designing optimal delivery or collection of routes from one or several depots to a number of customers subject to side constraints. Thus, the basic VRP can be described as vehicles that depart from the depot, visit one or more customers and return to the depot. The requirement of a solution for the VRP indicates clearly that the following objects should exist: depot, customers and vehicles.

According to our supply chain domain model, there exist a number of classes that represents the supply chain model. We should now super impose the VRP requirement onto the supply chain domain. It is clear that the VRP requires a simpler model for implementation. Thus the implementation requires a mapping and filter mechanism between the two models. Because the two models are well defined, the mapping can be seen as consistent across solutions. Note that the mapping is focused on the domain model and does not restrict the flexibility of the classes involved. The mapping considered here is only applicable to the final VRP routing algorithm and it is assumed that all required allocations and calculations have been done.

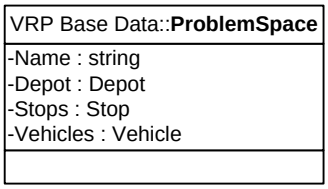

Figure 11: Problem Space Class

From the representative problem formulation, we define the base objects that are required as

- a list of stops,
- a depot,
- a list of vehicles (resources)

Vehicles do not form part of the original description according to Barbarosoglu and Ozgur (1999), but was identified as a critical enough side constraint input to be part of the base object model. The problem is to define an interface for the objects that is complete and sufficient enough to be used in the algorithm.

According to the component model, the base classes and cost functions have interfaces to the data objects, but the algorithm not. This extra level of abstraction insures that the algorithm operates on a known base that shifts the focus for the algorithm to the methods use and not data integration. The purpose of the base class component layer is to act as the interface between the data and the algorithm and provide addition storage structures required by the algorithm to execute efficiently.

3.3.3 Base classes (Solution Workspace)

The input data is loaded into data objects and then used through the problem space object. It consists of static data with all properties available that can assist in solving the problem. This optimization algorithm requires a known interface to work with. We define the base classes as interface for the data to the algorithm as well as algorithm specific storage area for specific algorithmic approaches.

The optimisation algorithm utilizes advance memory structures. The complexity of the base classes is dependent on the algorithm and will be discussed in the algorithm definition. The most important base class that is used is the SolutionWorkspace, which extends the ProblemSpace. The SolutionWorkspace implements RoutableStops and RoutableVehicles. These objects contain additional placeholders to allow for adaptiveness required by the cost and constraint functions. The solution workspace contains memory structures used in the algorithm.

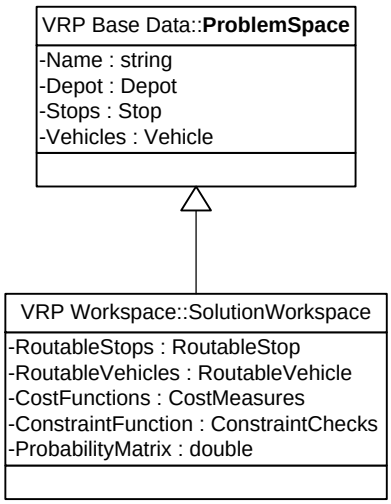

Figure 12: Solution Workspace

Any module using this VRP solution that implements the defined interfaces can provide data to the resulting algorithm.

3.3.4 Cost and constraint functions

This component represents the user defined objective functions as well as constraint functions. The implementation view a constraint function as *checking if a solution is valid*, while the cost function is the driver toward a good solution.

Implementing a crude cost function for any improvement heuristic will consist of calculating the compliance of the new solution to the constraints and then calculating the cost of the solution if no constraints have been violated. The solution provide for generic *StopData* per *RouteStop* to provide the cost function the ability to work from a cached solution base. See the Solomon implementation example cost function for a typical use of environment knowledge.

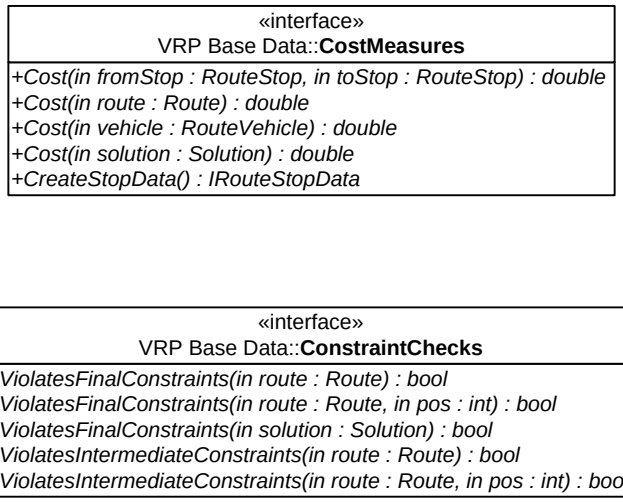

Figure 13: Cost and Constraint Interfaces

These interfaces allow the algorithm to call the appropriate cost or constraint function at the applicable time. The component design allows the cost function access to both the structured data objects as well as the data source. The interface to the data object is necessary for the cost function to obtain information required to calculate the result, e.g. in Solomon the cost function is a distance based function that requires the location of the stops to calculate the distance in Euclidian space. The Solomon implemented constraint function requires the time window of a stop to check the compatibility.

Both these functions required properties that relates directly to the base objects identified. This approach is sufficient for most VRP implementations. The interface to the data source implies access for the cost function to any data that is available in the system.

3.3.5 Optimization algorithm

The objective function of the problem is to minimize cost while adhering to all constraints. An adaptive object implementation allows for a higher level of abstraction of these functions, i.e. the 'user' of the algorithm has the ability to provide the cost and constraint functions. The design of the algorithm makes use of these external functions to guide the solution to a minimum.

The optimization algorithm has the base classes and cost functions as input. The base classes are well defined, but the cost functions are unknown for the purpose of this study. The goal is to solve the problem with this limited knowledge.

This paper will not discuss the multi-objective cost function, but the resulting solution should be easy to adapt to incorporate such cost drivers.

3.4 Implementation

The solution was developed on Visual Studio 2008, in the C# language. C# has the ability to implement interfaces, as do many other languages. The design is aimed to be acceptable in other languages such as C++ and Java as well. The use of interfaces is acceptable practice. It also utilizes the idea of delegates or function pointers. The use of delegates adds complexity to the code if it is not defined clearly.

An interface defines the communication boundary between two entities, such as a piece of software, a hardware device, or a user. It generally refers to an abstraction that an entity provides of itself to the outside. This separates the methods of external communication from internal operation, and allows it to be internally modified without affecting the way outside entities interact with it, as well as provide multiple abstractions of it. It may also provide a means of translation between entities which do not speak the same language, such as between a human and a computer. Because interfaces are a form of indirection, some additional overhead is incurred versus direct communication.

We utilize this concept to define methods outside of the algorithm to manipulate actions inside the algorithm. It is important to define the interfaces at the appropriate level. The 'user' has no knowledge of implementing an optimization algorithm, but determines the objective as well as constraints for the algorithm. The goal is to *abstract* these two elements of the algorithm to be easily controlled by the 'user'. These interfaces rely also on the base classes.

3.4.1 Constraints

The VRP, as in many real-world optimization problems, are solved subject to set of constraints. Constraints placed restrictions on the search space, specifying the regions of the space that are infeasible. Optimization algorithms have to find solutions that do not lie in infeasible regions. That is, solutions have to satisfy all specified constraints. The following types of constraints can be found:

- Boundary constraints, which basically define the borders of the search space. Upper and lower bounds on each dimension of the search space define the hypercube in which solutions must be found.
- Equality constraints specified that the function of the variables of the problem must be equal to a constant.
- Inequality constraints specified at the function of the variables must be less than or equal to a constant.

Feasible solutions can be compared by comparing the objective function value. Infeasible solutions are not that easy to compare. If the algorithm allows for an infeasible solution, it should be reflected in the cost function. The general notion is that an infeasible solution is much more expensive than a feasible solution. The allocation of cost to infeasible elements in the infeasible solution can however allow for tactical planning, where it might be better to remove the stop from the solution then servicing the stop at a high cost.

The proposed solution in this thesis requires the operator to implement its own cost and constraint functions. Investigation of more complex VRP problems resulted in similarity of constraint types. The aim is to provide the operator with guidelines on writing cost and constraint functions that integrates with the algorithm on a seamless manner. We identify constraints in the following areas:

- Relational constraints – this type of constraint depends **on** a relation between two objects. If split deliveries are not allowed, there must be a one to one relation between a vehicle and a stop. In the meta design, the user can indicate the cardinality of objects to enforce the constraint. The optimization algorithm can check these constraints whenever it is effective.
- Comparison constraint – this type of constraint depends on the aggregation of a property compared to a value. The value can belong to a relational object or it can be constant.
    - Relational object – Sum(Route.Stop.Volume) <= Vehicle.Volume
    - Constant – Count(Route.Stop) <= Solution.MaxStopPerRoute
- External constraint – we have to provide the user with the ability to add any type of constraint that cannot be modelled in our structure. This ensures 100% flexibility,

while the structured approach will be sufficient most of the time. The next paragraph discusses the use of interfaces for this kind of implementation.

3.4.2 Interfaces

We define a constraint function interface as a boolean function that tests the validity of the input object according to the defined constraint. The interface implements the constraint for all base objects as input parameter. This allows the optimization algorithm to call constraint checks whenever the appropriate input parameter has been changed and require a validity check. It also provides the user the ability to determine the granularity of checks, i.e. a route can be checked for validity, or a solution can be checked for validity by checking all elements. Using this level will cause the algorithm to be slow, but it will be an accepted method. This underline the ability of the methodology to solve 'any' problem, but emphasize the importance of providing clear guidelines on writing interfaces to ensure most efficient implementation of cost and constraint functions.

We define a cost function interface as a function that returns a value between zero and infinity. The algorithm will use this value as indication of the quality of the solution. The VRP solution is the minimization of the cost function.

3.5 Base classes

The previous paragraph emphasizes the definition of the base classes as tool in the solution. From the problem formulation it is clear that the base classes interface to the data objects, the cost functions and the optimization algorithm. The subsequent paragraphs offer an overview of how these interfaced layers influence the design of the base classes.

This research is also done to initiate a different thought process around the VRP and its approach. Base classes consist of more abstract meaning. This will allow further research in applying the solution in similar problems by mapping the implementation to the abstract concepts.

A typical implementation for Solomon problems can be defined as follows:

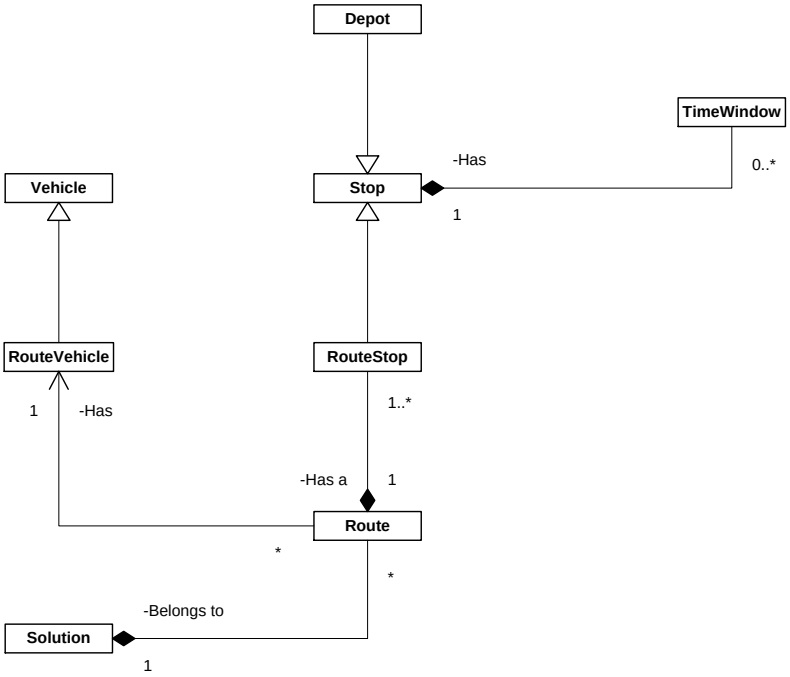

Figure 14: Basic domain class relations

3.6 Data objects

The data objects is defined as the user's objects that comprise of information available for the problem space. Our object approach is based on the ESI implementation which maps physical data to objects through an object service. This service is controlled through meta-data.

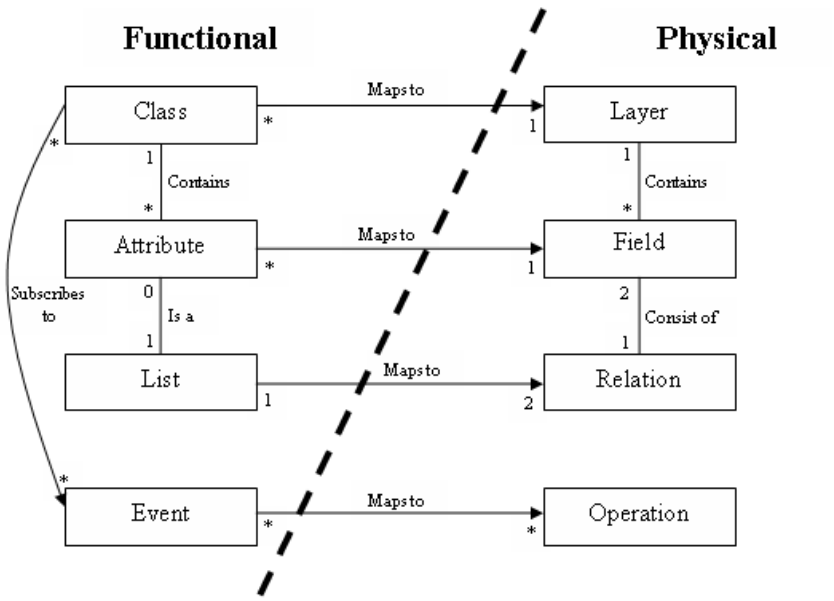

Figure 15: Meta mapping, functional to physical

This implementation is too flexible as it allows the user to define any class of any type. We introduce a domain structure into the ESI that functions similar to an abstract class layer.

The ultimate goal is not to solve benchmark problems, but to apply the algorithm in the real world. The data objects represent the real word objects in a supply chain implementation. The following objects are identified in the supply chain configuration:

3.7 Cost and constraint functions

The base classes provide feedback from the algorithm to the cost function. The algorithm store information in the base class structure and any request for a cost calculation is instantiate through a base class. This emphasizes the importance of the base class definition to support a scaled approach when calling a cost function to minimize processing power. The aim is to enable calculation to be done on partial knowledge through calculating the change in value, for example, if two stops are swapped between routes, we would like to know the net effect on the current solution by just calculation the affected routes.

3.7.1.1 Solomon Function

The Solomon benchmark problems implement a stop with a cost function as the Euclidian distance of the routes that comply with the constraint functions. Solomon constraints consist

of one time window and a vehicle capacity determined by the homogeneous fleet. The Solomon cost and constraint functions can be implemented as follows:

- Capacity – the addition or removal of a stop influence the capacity of a route. The action is not sensitive on the position in the route.
- Calculate time – the time constraint check calculates the validity from the previous stop and influence all subsequent stops' validity.
- Distance – the addition or removal of a stop influence the distance of the route, but can be re-calculated through the previous and next stop

From the above properties we can define the following optimization rules:

- Decide on the order of calculation of constraints. An easier and quicker calculation that can violate the constraint should be done first to prevent unnecessary use of processing power.
- The storage of calculated data should be planned and maintained to assist in reducing processing.

3.7.1.2 Peak and off-peak travel times

Adapting the basic Solomon problem to allow for peak and off-peak travel times can be isolated to the cost and constraint component. Since the cost function has access to the data source, the implementation consists of a multi-dimensional matrix with origin to destination travel times per time of day.

The implementation of the optimisation algorithm for the benchmark Solomon problems can rely on a consistent cost between two consecutive stops as distance and time travelled will always be the same. The implementation of peak and off-peak time matrices results in sudden change of cost, depending on the position of two consecutive stops in a route. We can identify the following three scenarios as possible results of the peak and off-peak travel-time:

1. The cost is too high for the ideal solution in both peak and off-peak time, which will result in the combination of stops not evaluated in the algorithm.
2. The cost is low enough for the ideal solution in both peak and off-peak time, which will result in the combination of stops being evaluated in the algorithm.

3. The cost is low in off-peak and too high in the peak time, which results in the stops being favourable in some routes and not in other similar looking routes.

Since the cost and optimisation algorithm is removed from each other, the algorithm had no domain knowledge to work on from the start, but rely on analysis of the environment and feedback from the results achieved. The pheromone trial will indicate that the combination is low in scenario 1, high in 2 and high in 3 because the low cost part of the peak and off-peak time is part of the possible best solution.

3.8 Optimization algorithm

The optimization algorithm require as much as possible information to assist in making efficient moves. In our approach, the algorithm does not communicate with the object data. All it can see it is the defined base classes and functions. The base classes must therefore be designed to assist the algorithm.

The algorithm depends on the base classes to make sense of the state of the structure and keep track of combinations of stops and vehicles. The structures that serve as input for the algorithm and is fixed. The object model defined additional structures for the use of the algorithm. These structures include information that allows the AMP (adaptive memory programming) to be implemented as part of the algorithm.

The rest of the thesis will focus on the design of the algorithm that employs the structures to assist in effective execution.

3.9 Summary

Integration of systems between cooperating enterprises or between different departments is most important because it provides highest value. Providing a theoretical sound solution that has the agility to adapt to business needs is within reach through the use an adaptive object model.

Deploying a data shy algorithm in the complex VRP environment is no small task. This chapter defines the base structures necessary to implement the supply chain domain. It describes the different components and their relationships in the building blocks towards an integrated solution.

Academic collaboration with science-based industry provides an occasion to consider underlying differences between academic and industrial science when only their ends, theories vs. products, distinguish them. Industry's relative indifference to theory nudges academic collaborators toward speculation. Industry entices academics to know less about more. The industry requirement should not derail the scientific effort applied in this thesis.

The next step is to design the algorithm that can deal with all the requirements, which still include the traditional optimisation goals.

4 ANT SYSTEM ON ADAPTIVE OBJECTS ALGORITHM

4.1 Approach

Solving the vehicle routing problem in its basic format is already an NP-hard problem. Exact methods have proved to be inefficient and time-consuming in trying to solve this problem. Previous attempts on solving the VRP have indicated that heuristic methods result in the best feasible solution in an acceptable time. When we add additional constraints to the basic VRP, we increase the difficulty of the solution exponentially. We must also consider the size of the data set that needs to be optimized.

Heuristic methods search only parts of the solution space. This result in the quicker results of the algorithm, but does not guarantee a best solution. Previous results have shown that heuristic methods can achieve optimal or near optimal results repeatedly. The meta-heuristic method has a guidance procedure of some sort to help it traversing through the solution space.

The guidance procedure is dependent on the type of heuristic selected for the solution, as well as additional knowledge from the problems space implemented by the algorithm. This additional information about the problem beforehand can assist the algorithm in more effective search paths. A meta-heuristic is the implementation of a heuristic method with a guidance procedure.

A hyper-heuristic is a heuristic search method that seeks to automate, often by the incorporation of machine learning techniques, the process of selecting, combining, generating or adapting several simpler heuristics (or components of such heuristics) to efficiently solve computational search problems. Hyper-heuristics can be thought of as "heuristics to choose heuristics". One of the motivations for studying hyper-heuristics is to build systems which can handle classes of problems rather than solving just one problem.

The fundamental difference between meta-heuristics and hyper-heuristics is that most implementations of meta-heuristics search within a search space of problem solutions, whereas hyper-heuristics always search within a search space of heuristics. Thus, when using hyper-heuristics, we are attempting to find the right method or sequence of heuristics in a

given situation rather than trying to solve a problem directly. Moreover, we are searching for a generally applicable methodology rather than solving a single problem instance.

Memetic Algorithms (MA) is used as a synergy of evolutionary or any population-based approach with separate individual learning or local improvement procedures for problem search. The concept of a meme (a unit of ideas that can be transmitted from one item to another) relates to the pheromone values that can be copied between ants. The combination with a meta-heuristic creates powerful decision making ability that can guide the algorithm through the solution space.

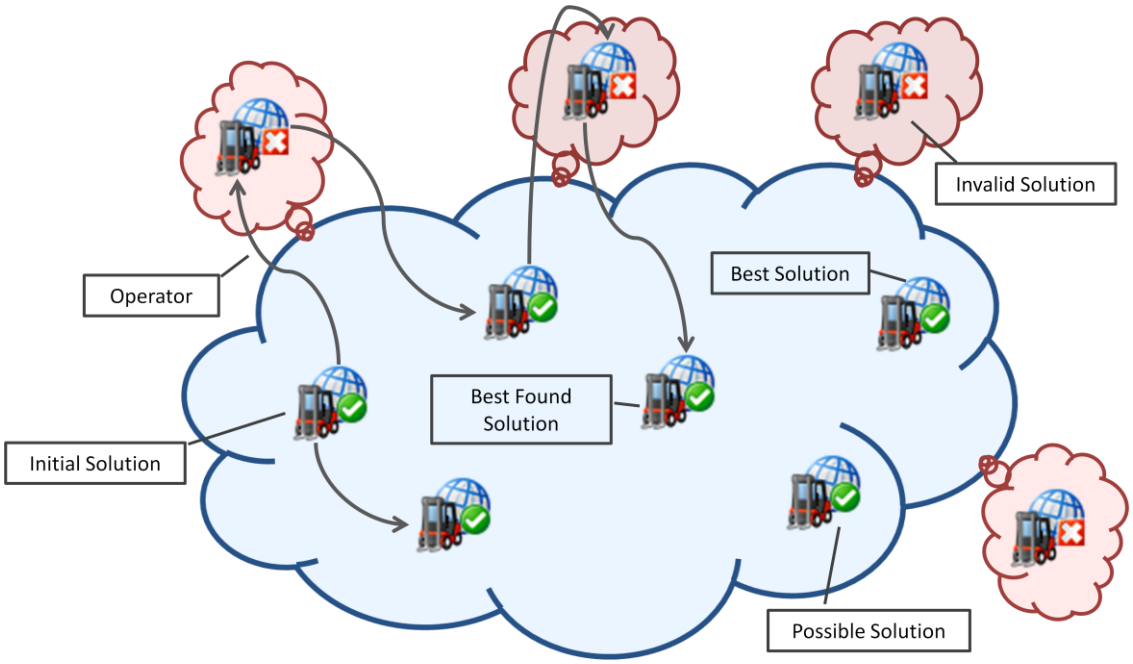

Figure 16: Solution Space

Figure 16 explains the methodology of heuristic methods for solving the particular problem. The solution space consists of all possible solutions for the specific problem. Theoretically we can develop an algorithm that has the ability to generate all of the possible solutions such as branch and bound methods. As we have already seen, this method will take an eternity on the complex problem that we are trying to solve. A meta-heuristic can search effectively through the solution space. The algorithm can allow invalid solutions which might lead to better valid solutions. The algorithm might not reach the best solution in the allowed time.

Let S be a set of solutions to a particular problem, and let f be a cost function that measures the quality of each solution in S. The neighbourhood N(s) of a solution s in S is defined as the

set of solutions which can be obtained from s by performing simple modifications. Roughly speaking, a local search algorithm starts off with an initial solution in S and then continually tries to find better solutions by searching neighbourhoods. A local search process can be viewed as a walk in a directed graph G=(S,A) where the vertex set S is the set of solutions and there is an arc (s,s') in A if and only if s' is in N(s). By considering the cost function as an altitude, one gets a topology on G=(S,A).

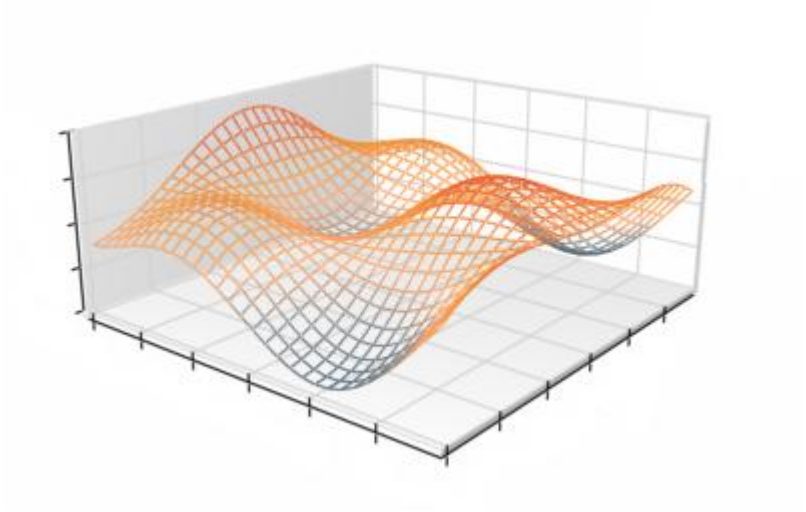

Figure 17: Solution Neighbourhood

The efficiency of a local search method depends mostly on the modelling. A fine-tuning of parameters will never balance a bad definition of the solution set, of the neighbourhood, or of the cost function.

The topology induced by the cost function on G=(S,A) should not be too flat. The cost function can be considered as an altitude, and it therefore induces a topology on G=(S,A) with mountains, valleys and plateaus. It is difficult for a local search to escape from large plateaus since any solution that is not in the boarder of such a plateau has the same cost value as its neighbours, and it is therefore impossible to guide the search towards an optimal solution. A common way to avoid this kind of topology on G=(S,A) is to add a component to the cost function which discriminates between solutions having the same value according to the original cost function.

When the problem is known beforehand, we can predict the surface of the graph, and adapt the algorithm accordingly. The problem definition in this study has a basic knowledge of the

class of problems, i.e. VRP. The cost and constraint functions determine the surface of the chart, but are not known at all at the start of the algorithm. This leads to an evolutionary process to detect the surface type during the execution of the algorithm.

Our evolutionary meta-heuristic makes use of the well-known two-stage and multi-start local search (MLS) frameworks. In two-stage framework the initial solution created in the first stage is subsequently improved in the second one. Because the environment is unknown, the multi-start construction heuristics assist in an immediate evaluation of the environment.

In the first stage we generate an initial solution with the help of a construction heuristic. There exist several methods and we make use of the sequential insertion heuristic (SIH), random-best selection for constructive ant path and best accept for constructive ant path. These methods result in solutions that are feasible but not necessarily the best. The feasibility of the solution ensures that it existing our solution space (see the initial solution in Figure 16).

The initial evaluation includes statistical analyses of the best-case cost between nodes. Memory structures are initialised with knowledge gained from the initial solution in combination with the analysis. The methods used in the improvement stage are fixed and we can claim to already know how the improvement heuristic works. Now we aim to align the information and actions in the construction phase to provide a result that will assist in quicker convergence.

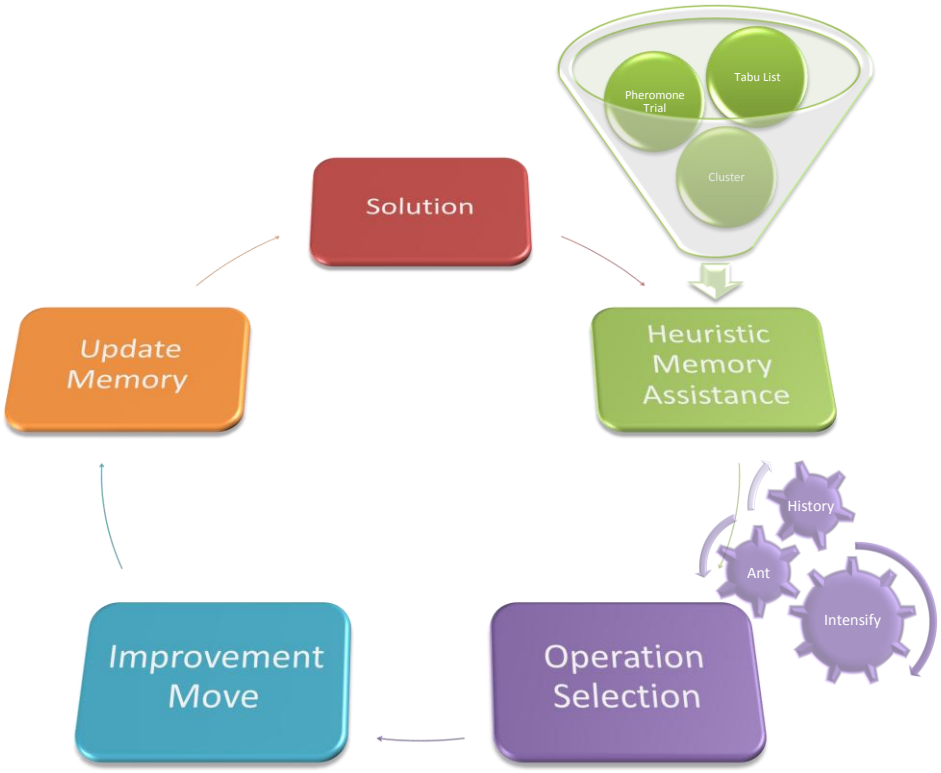

Figure 18: Solution approach overview

The improvement stage traverses from the current solution to a neighbour solution. The move generates a new solution which might have been created previously from other combinations of moves on other solutions. This can result in cycles in our search path, which leads to revisiting existing solutions and result in unnecessary computational time. One of the objectives will be to prevent such cycling. After a specified number of iterations the algorithm has visited a number of solutions from which the best solution is kept. The solution is not necessarily the best solution for the problem, but represents the best-visited solution. Our goal is to guide the search path in such a way that we cover as wide as possible area of the solution space.

From Figure 16 we can see that the path to the best solution might have to go through a 'not so good solution' or even an invalid solution before the best solution is reached. Operations applied on a solution can result in a not feasible solution. We can consider this as a stepping-stone towards the next solution, or it can be seen as a waste of computational time.

The improvement phase implementation is based on an Ant Colony System and a local Tabu Search Method. Tabu search has a rationale that is transparent and natural: its goal is to emulate intelligent uses of memory, particularly for exploiting structure. Since we are creatures of memory ourselves, who use a variety of memory functions to help thread our way through a maze of problem-solving considerations, it would seem reasonable to try to endow our solution methods with similar capabilities.

The Ant Colony System (ACS) algorithm is based on a computational paradigm inspired by the way real ant colonies function. The medium used by ants to communicate information regarding shortest paths to food, consists of pheromone trails. A moving ant lays some pheromone on the ground, thus making a path by a trail of this substance. While an isolated ant moves practically at random, an ant encountering a previously laid trail can detect it and decide, with high probability, to follow it, thus reinforcing the trail with its own pheromone.

The collective behaviour that emerges is a form of autocatalytic process where the more the ants follow a trail, the more attractive that trail becomes to be followed. The process is thus characterized by a positive feedback loop, where the probability with which an ant chooses a path increases with the number of ants that previously chose the same path. The ACS paradigm is inspired by this process. In this traditional each ant generates a solution probabilistically or pseudo-probabilistically based on the current pheromone trail. Each iteration constructs a new solution utilizing the information in the pheromone trial. To prevent cycles and improve efficiency, the study introduces several new memory mechanisms which will guide the ants.

The following sections discuss the specific methods used to traverse through the solution space in more detail. It will also point out where knowledge about the problem beforehand can have an effect on the implementation of the solution. The algorithm consists of the following steps:

```
Evaluate the environment.
Construct a cost and compatibility matrix.
Build clusters.
Construct initial solutions
Deduct information from first round optimizations.
Setup memory structures for improvement phase
Apply the algorithm.
Run thread for each solution.
Determine state of solution and algorithm to morph to next phase.
Update global memory structures.
```

Algorithm 2: High Level Approach

The following paragraphs will discuss the building blocks separate and then conclude with the system.

4.2 Compatibility and Cost Matrices.

The construction of initial environment evaluation structures requires some knowledge of the constraint and costs that is applicable on the problem. The purpose of the compatibility and cost matrices is to provide a structure with a known access time which ensures that the initial setup can be done in a determined time. The behaviour of the cost and constraints is not known to the algorithm, and we assume a best case scenario at this stage. A definite result from this calculation is that any incompatible combination will never be compatible, but a compatible combination does not necessarily ensure the compatibility.

4.2.1 The cost matrix

Cost is the most important calculation in the optimization of the VRP. The goal of the study is to minimize cost, which requires each visited solution to calculate the cost for comparison. The adaptive object model approach hides the knowledge about the cost function from the algorithm and thus we cannot predict the factors that contribute to the cost of a solution. A

nested memory approach is followed where we try and exclude unnecessary solutions, moves, etc.

Predicting the running time of the algorithm is complicated by the encapsulation of the cost function due to the adaptive approach. The cost function has access to all information from the base data as well as the current solution on route and total solution level. The calculation can become quite complex when additional attributes make up the information at a stop and is used in the function. The cost matrix hides the complexity to assist in a determined cost function time for an intermediate level of solution. This intermediate level presents an approximation of the surface of the chart shown in Figure 17.

To assist in this approach, we implement a best-case cost matrix that is used on a high level to guide the algorithm away from very poor solutions. It also gives the advantage of getting the algorithm quick out of the blocks with a memory structure already setup. The cost matrix is calculated as the lowest possible cost between two stops, which represent the best situation the stops can be in relation to each other. The resulted cost matrix is used as input for the initial solution clustering as well as the compatibility matrix. Costs range from 0 to infinity and cannot be negative.

4.2.2 The compatibility matrix

The compatibility matrix acts on the cost between nodes and will be carried over to the improvement stage as a permanent tabu list that is build up from environment knowledge as well as basic calculations. The matrix is maintained through the iterations per specific solution.

The values range from 0 to 1. The 0 value represents no possibility; whilst 1 represent a good probability. A 0 value should represent no possibility ever and will be carried over to other compatibility matrices for other solutions. The information is used in calculating the probability of a move to create new routes. The compatibility matrix reduces the number of calculation possible for a move evaluation and thus allows for more possible moves in the finite iterations. One of the well known contributors to the matrix is the Time Window Compatibility which is described in detail in the next section.

The number of impossible combinations between stops assists to reveal the distribution of solutions in the solution space. This information form a basis of where possible mountains exist on the surface of the graph G(S,A). The compatibility matrix values are deducted from cost which is normalized on the global worst and best cost. The probability of the decision variable x_{ij} is defined by the linear position between the minimum and maximum cost from the stop. The probability is then factored to the number of possible neighbours from and to the stop.

$$P(x_{ij}) = 1 - \left(\frac{\max(c_{ik}, c_{ki}) - c_{ij}}{\max(c_{ik}, c_{ki}) - \min(c_{ik}, c_{ki})}\right) \times \frac{\sum_0^n x_{ik} + \sum_0^n x_{ki}}{2n}$$

Equation 2: Probability of neighbouring stops on environment

The values of the probability are proportional to the cost when there are no constraints in the problem. If a stop has only a few possible neighbours in relation to the total number of stops, the probability will be reduce because of the last term in the formula. The purpose is to reduce the overall probability of the stop, i.e. the algorithm does not need to include the stop in decisions regularly because the likelihood that the best stop has already been selected is quite high. The probability matrix assists in improving computational time.

The following paragraph explains the influence of time windows on the probability matrix and it is clear to see that it has a major computational cost saving effect. The reader must keep in mind that the sample is an extract of a known domain, whilst the algorithm in this study will never know about such domain specific information. The cost function between two stops will reflect the information and that is deemed to be sufficient for building the memory structure discussed.

4.2.2.1 Time Window Compatibility

Time Window Compatibility (TWC) refers to the compatibility of the time window(s) of one stop with regards to another. A good TWC figure indicates that the two nodes are likely to be inserted in sequence on the same route. In many cases two customers can be located next to each other, but their time windows are not compatible. The trade-off between distance (i.e. cost) and time (i.e. customer delight) is an inherent part of the problem.

Insertion of stops in a heuristic fashion requires a selection process that result in a possible next stop. The TWC can assist us in ruling out infeasible stops from the start. If we know that a stop is not a neighbour of the current stop, we do not even waste time of trying to implement that stop as a next stop. The neighbours of a stop are made up of all the time window compatible stops. We utilise the TWC principle as proposed by Van Schalkwyk (2002) to explain the function of the probability matrix.

The figure below illustrates a scenario where we evaluate the time adjacency of node *i* and node *j*. This scenario assumes that there will be a definite overlap in time windows between the two nodes. Other scenarios will subsequently be discussed.

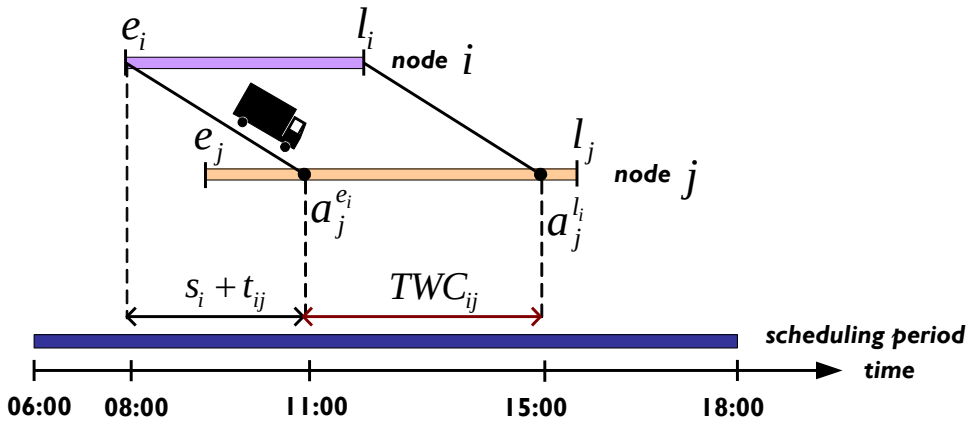

Figure 19: The basic TWC calculation - Scenario 0

Scenario 0: $if\ a_j^{e_j} > e_j\ and\ a_j^{l_i} < l_j$

Customer i specified a time window (e_i, l_i) between 8:00 and 12:00, and customer j requires service between 9:00 and 16:00 (e_j, l_j). If serviced started at node i at e_i (the earliest feasible time), its arrival at j would be:

$$a_j^{e_i} = e_i + s_i + t_{ij}$$

In this scenario equal 11:00.

Similarly, al would be the arrival at j if service started at node i at the latest possible time (l_i):

$$a_j^{l_i} = l_i + s_i + t_{ij}$$

In this scenario equal 15:00.

The difference between $a_j^{e_i}$ and $a_j^{l_i}$ will yield the amount of time overlap between i and j:

$$TWC_{ij} = a_j^{l_i} - a_j^{e_i}$$

In this scenario it equals 4 hours. The significance of this value is that the bigger the overlap, the better we can insert the two nodes in a sequence. This also ensures that the customer with a big overlap has higher probability and can be used during the optimisation phase more regularly because of the better possibility of a fit.

A number of different scenarios will be illustrated in the following figures.

Scenario 1: If $a_j^{l_i} > l_j$

If the earliest arrival time at node *j* is inside the acceptable time window, but the latest arrival time is outside of the acceptable time window of node *j*, the two customers only partly overlap. The TWC_{ij} is then calculated by the following equation:

$$TWC_{ij} = l_j - a_j^{e_i}$$

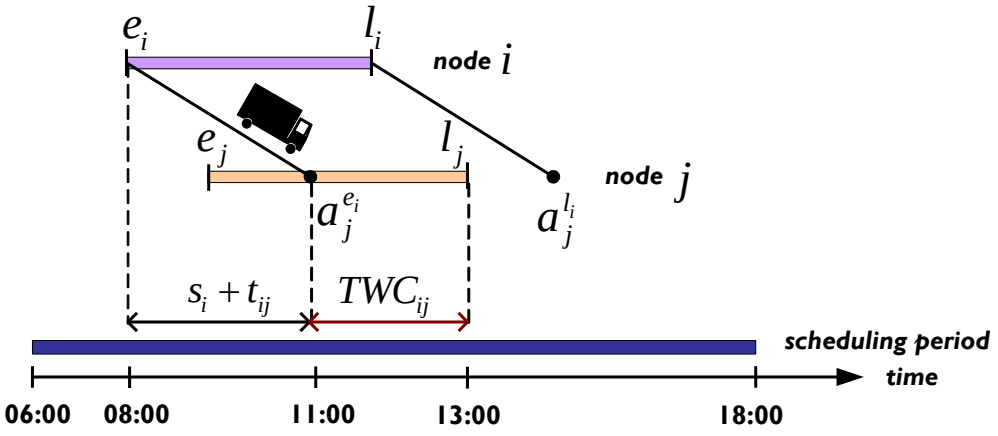

Figure 20: Scenario 1 TWC calculation

Scenario 2: If $a_j^{e_i} < e_j$

If the vehicle arrives at the earliest feasible time and this is before the acceptable time window of node j, and the arrival of the latest feasible time at node *j* is inside the acceptable time window, the two customers only partly overlap. The vehicle has to wait to service customer j. The TWC_{ij} is then calculated by the following equation:

$$TWC_{ij} = a_j^{l_i} - e_j$$

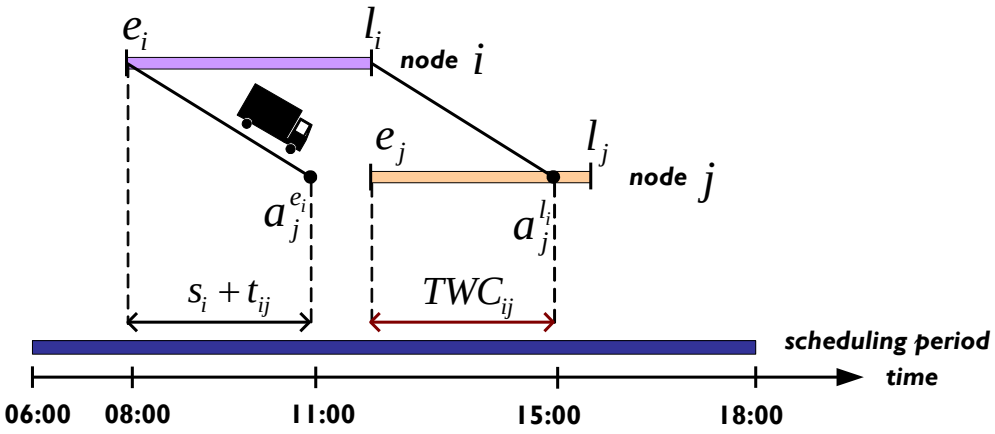

Figure 21: Scenario 2 TWC calculation

Scenario 3: If $a_j^{e_i} < e_j$ and $a_j^{l_i} < e_j$

If the latest arrival time at node *j* is earlier than the start of the acceptable time window at node *j*, the vehicle always waits at node j, irrespective of the arrival time at node i. The arrival at j is always before its acceptable start time. This value will be negative, and calculated as follows:

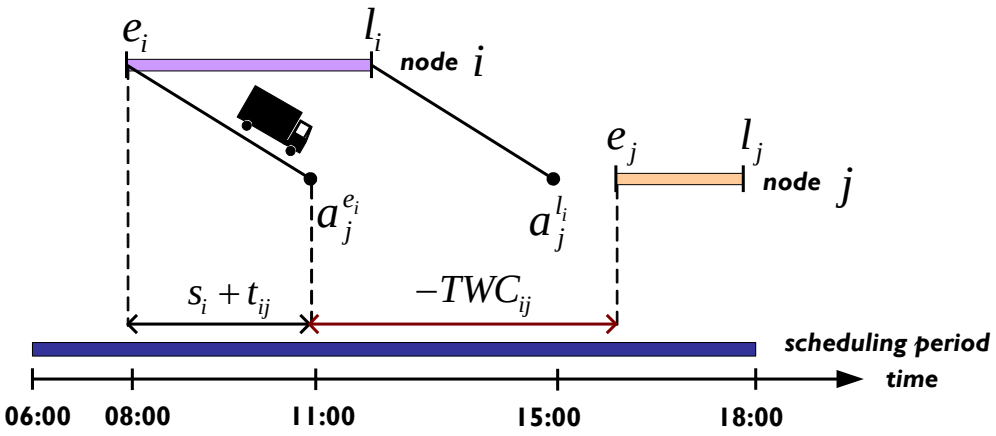

Figure 22: Scenario 3 TWC calculation

Scenario 4: If $a_e > l_j$ and $a_l > l_j$

If the arrival time at j is always bigger than the latest acceptable time at j, the node-combination is infeasible. The nodes forming part of this combination will typically be eliminated before starting the algorithm, as they can obviously not be included in the current route under construction.

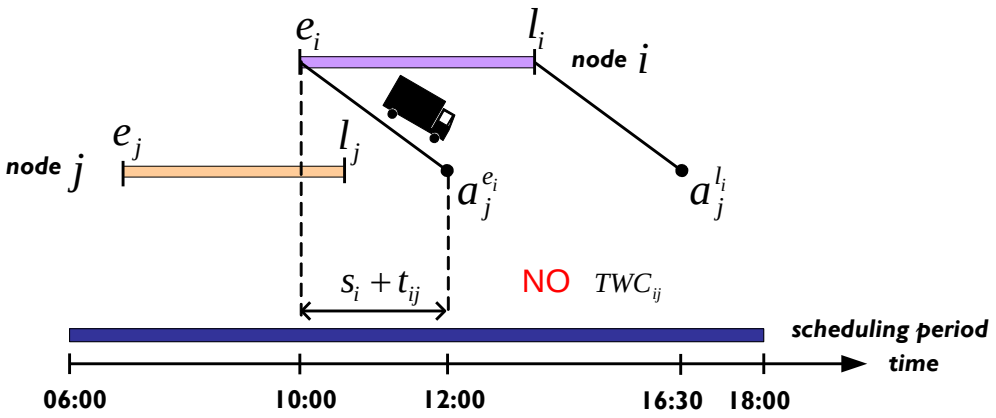

Figure 23: Scenario 4 - infeasible combination

4.3 Clustering assistance in probability

The problem formulation in Chapter 2 has referred to clustering as tool in assisting solving the complex problem. Cluster analysis has been identified as a core task in data mining. The VRP problem has much less entities to cluster than normal data mining problems. Clustering can be used to the benefit of the solution, without adding too much calculation overhead. Finding a good set of clusters, each one comprising several customer sites, without relying on routing information is a quite difficult task.

The top-down view regards clustering as the segmentation of a heterogeneous population into a number of more homogeneous subgroups. A bottom-up view defines clustering as finding groups in a data set 'by some natural criterion of similarity'. There are others who believe that the fundamental question is if two items are not in the same cluster. Defining the similarity between stops depend on the combination of possible routes in the solution, which is dependent on cost relation between the stops as well as constraints on the combinations.

The number and the extent of the clusters built by a clustering algorithm generally depend on a set of parameters that can be tuned in one way or another. But this possibility is implicitly limited by the similarity measure used for comparing the elements to cluster. We require a cost function between entities to indicate the similarity measure. We also consider several methods of clustering and argue that the computational time is deterministic ant thus measureable to ensure that the computational effort is worth it.

4.3.1 Review of clustering methods

Clustering is the process of grouping the data into classes or clusters so that objects within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other cluster. Also, the process of grouping a set of physical or abstract objects into classes of similar objects is another definition. Clustering can be used to gain knowledge of a data set or used as a pre-processing technique for classification. This is the primary goal for the algorithm in this study to align with the adaptive object model approach.

4.3.1.1 Partitioning methods

This is the most simple of the clustering methods. This categorization could be too broad because Clustering is equivalent to Partitioning. The best known method is k-means. It is a

basic clustering algorithm that creates circular clusters in 2D and spherical clusters in 3D. It creates k clusters centred on a centroid. The centroid is almost always an artificial point. This algorithm is extremely sensitive to outliers that are a significant distance away from an actually perceived cluster.

4.3.1.2 Hierarchical Method

Hierarchical methods group data into a tree of clusters. There are two basic varieties of Hierarchical algorithms; agglomerative and divisive. Agglomerative clustering is a bottom up strategy where we start at individual data object each in its own group. Then we combine the nearest pair of object into a new group. This process of combining closest groups continues until we have all the objects in one cluster. Divisive clustering proceeds in the same way except that we start out with all data object in one cluster and then we end with all objects in separate clusters.

4.3.1.3 Density Methods

Density clustering methods are very useful of accurately finding clusters of any shape giving the correct (yet hard to determine parameters). Density is obviously determined by how many data object are contained within a certain space of the dataset. This means that we need to map the data to some sort of graph.

4.3.1.4 Model Methods

Conceptual clustering is a form of clustering in AI that given a set of unlabeled objects, produces a classification scheme over those objects. Really only works on a specific kind of data and a model is formed to cluster that kind of data.

4.3.2 Clusters and data environment

Dondo and Cerdá (2006) use a three-phase heuristic algorithmic approach for the multi-depot routing problem. The proposed clustering algorithm that exploits time-window constraints to generate feasible clusters seems to work well even for R-class problems. The three-phase hybrid approach is as robust as the optimization methods and capable of solving problems with 100 nodes at reasonable solution time. Numerical results indicate that the cluster-based optimization method proved to be quite successful on a variety of Solomon's single depot homogeneous-fleet benchmark problems and new multi-depot heterogeneous fleet VRPTW

instances introduced in their paper. Optimal or near optimal solutions were obtained for a significant number of C-class problems of different sizes. For RC and R-class problems, the sub-optimal gap increases but it remains within acceptable limits.

This research follows a similar approach. The aim is to gather enough information that does not limit the improvement algorithm in making decision and also contain aggregated information to reduce variables and improve speed.

The two cluster techniques used is partitioning and density clustering. The subsequent paragraphs describe the use of specific methods.

4.3.3 Cluster Methods

4.3.3.1 DBSCAN

The key idea of the DBSCAN (Moreira, Santos and Carneiro, 2005) algorithm is that, for each point of a cluster, the neighbourhood of a given radius has to contain at least a minimum number of points, that is, the density in the neighbourhood has to exceed some predefined threshold. This algorithm needs three input parameters:

- k, the neighbour list size;
- Eps, the radius that delimitate the neighbourhood area of a point (Epsneighbourhood);
- MinPts, the minimum number of points that must exist in the Eps-neighbourhood.

The clustering process is based on the classification of the points in the dataset as core

points, border points and noise points, and on the use of density relations between points (directly density-reachable, density-reachable, density-connected [Ester1996]) to form the clusters.

```
For each stop, calc all neighbours and sort
Calculate the average distance between a stop and its closest
neighbour
Identify the Directly Density-Reachable (DDR) points, i.e. points
closer than the threshold Epsilon.
Create clusters for DDR Points
```

Algorithm 3: DBSCAN

We replace the concept of distance to that of cost. The DBSCAN algorithm depends on the calculation of a centroid for a cluster. This is traditionally calculated as the average x and y in a Euclidian space. The concept of location exist in the VRP, but the addition of constraints and cost functions complicates the relationship between stops and thus cannot be deemed as efficient parameter to use as input for the clustering algorithm. The Shared Nearest Neighbour algorithm does not depend on a centroid for a cluster.

4.3.3.2 Shared Nearest Neighbour algorithm

The SNN algorithm (Ertöz, Steinbach and Kumar, 2003), as DBSCAN, is a density-based clustering algorithm. The main difference between this algorithm and DBSCAN is that it defines the similarity between points by looking at the *number* of nearest neighbours that two points share. Using this similarity measure in the SNN algorithm, the density is defined as the sum of the similarities of the nearest neighbours of a point. Points with high density become core points, while points with low density represent noise points. All remainder points that are strongly similar to a specific core points will represent a new clusters.

The SNN algorithm needs three inputs parameters:

- K, the neighbours' list size;
- Eps, the threshold density;
- MinPts, the threshold that define the core points.

After defining the input parameters, the SNN algorithm first finds the K nearest neighbours of each point of the dataset. Then the similarity between pairs of points is calculated in terms of how many nearest neighbours the two points share. Using this similarity measure, the density of each point can be calculated as being the numbers of neighbours with which the number of shared neighbours is equal or greater than Eps (density threshold). Next, the points are classified as being core points, if the density of the point is equal or greater than MinPts (core point threshold). At this point, the algorithm has all the information needed to start to build the clusters. Those start to be constructed around the core points. However, these clusters do not contain all points. They contain only points that come from regions of relatively uniform density. The points that are not classified into any cluster are classified as noise points.

```
Identify the k nearest neighbours for each point
Calculate SNN similarity between points
Calculate SNN density of each point
Detect core points
Form cluster from core points
Identify noise points
Assign the remainder points to the cluster that contains the most
similar core point
```

Algorithm 4: SNN

4.3.3.3 k-Means

k-Means is a basic partitioning clustering algorithm that creates circular clusters in 2D and spherical clusters in 3D. It creates *k* clusters centred on a centroid. This centroid is almost always an artificial point. This algorithm is extremely sensitive to outliers that are a significant distance away from an actually perceived cluster.

4.3.3.4 k-Medoids

The k-Medoids algorithm (Park, Lee and Jun, n.d.) is very similar to k-Means with the small exception of instead of creating an artificial point to recalculate the mean point, k-Medoids recalculates from the nearest actual point in a data set. The reason for this is that it is not acceptable to outliers that are extremely far away from it. A very large problem of k-Medoids is that it doesn't scale very well at all. It does however fit in well with the use of the encapsulated cost function in our adaptive approach. K-medoid clustering algorithm is as follows:

```
The algorithm begins with arbitrary selection of the k objects as
medoid points out of n data points (n>k)
After selection of the k medoid points, associate each data object
in the given data set to most similar medoid. The similarity here
is defined using distance measure that can be Euclidean distance,
Manhattan distance or minkowski distance. We translate distance as
a cost function.
Randomly select nonmedoid object O'
compute total cost , S of swapping initial medoid object to O'
If S<0, then swap initial medoid with the new one ( if S<0 then
there will be new set of medoids)
repeat steps 2 to 5 until there is no change in the medoid.
```

Algorithm 5: k-Medoid

4.3.4 Cluster implementation

Data mining in general is the search for hidden patterns that may exist in large databases. To the data environment preparation task at hand, the attractiveness of cluster analysis is its ability to find structures. Using the described clustering methods on an unknown data environment is insufficient if viewed in isolation. The density clustering methods of DBSCAN and SNN is possible in a clustered or random cluster environment. In a random environment, all points are either seen as noise, or the result is one cluster. Partitioning methods such as K-means and K-medoids will always provide the same number of clusters. The challenge is to determine k for the problem environment.

The SNN method uses the neighbour count to determine cluster density. This method is easy to convert to work with the abstract cost functions. It can also handle clusters of different densities in one problem environment. This can be applicable to scenarios where stops have a high density close to the depot, and a lower density further away. DBSCAN would not be able to cluster the lower density areas.

The subsequent paragraphs provide a brief overview of typical results in the density clustering environment. The study approaches the density clustering as the most significant method. We argue that if a sufficient cluster solution is found through this method, the initial solution can produce a close to final solution combination of stops and the selection of subsequent improvement operations can be set to focus on segment combinations instead of stop relations.

The Solomon C class problems are good examples and are discussed in the results section to show the effect. The results also indicate the positive effect for other types of problems, even on random dispersed problem spaces.

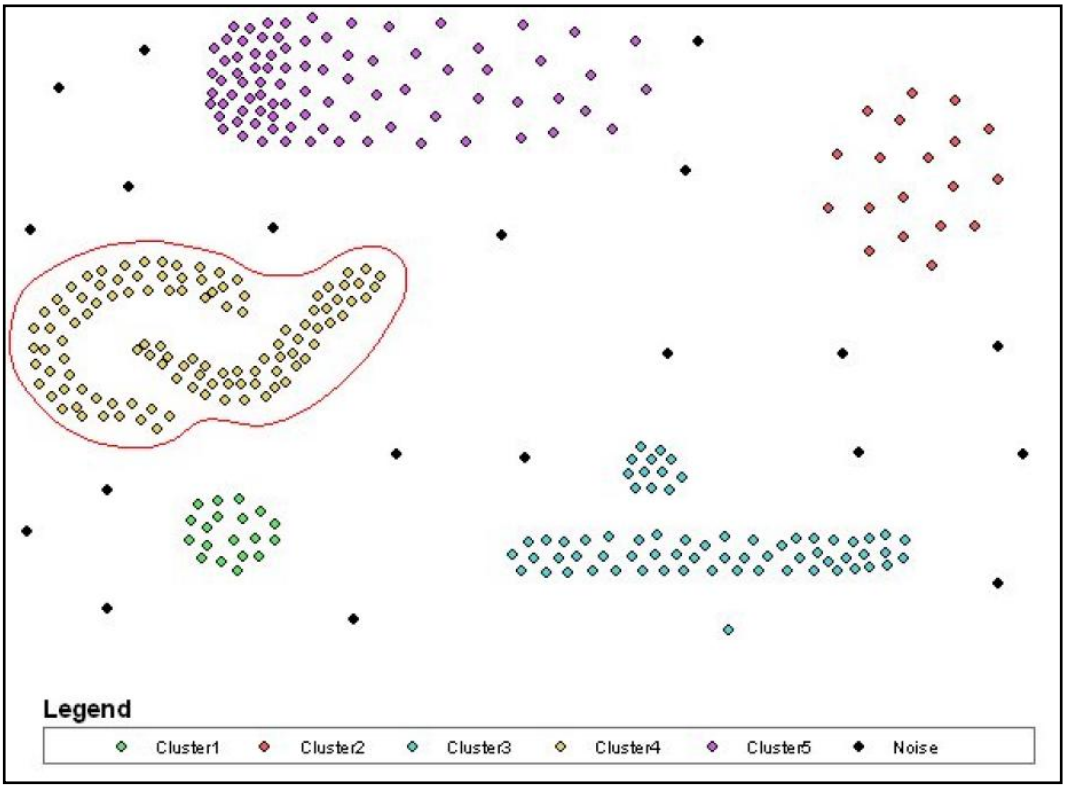

Figure 24: DBSCAN cluster

Figure 24 indicates the result of DBSCAN algorithm on a benchmark set op points. Cluster 4 can also be viewed as 2 clusters. Because of the definition of DBSCAN and the density setting, cluster 4 is seen as one cluster. The density was set low enough to incorporate cluster 2. The SNN method can be seen as determining its own local cluster density. We can see the 7 clusters with different densities.

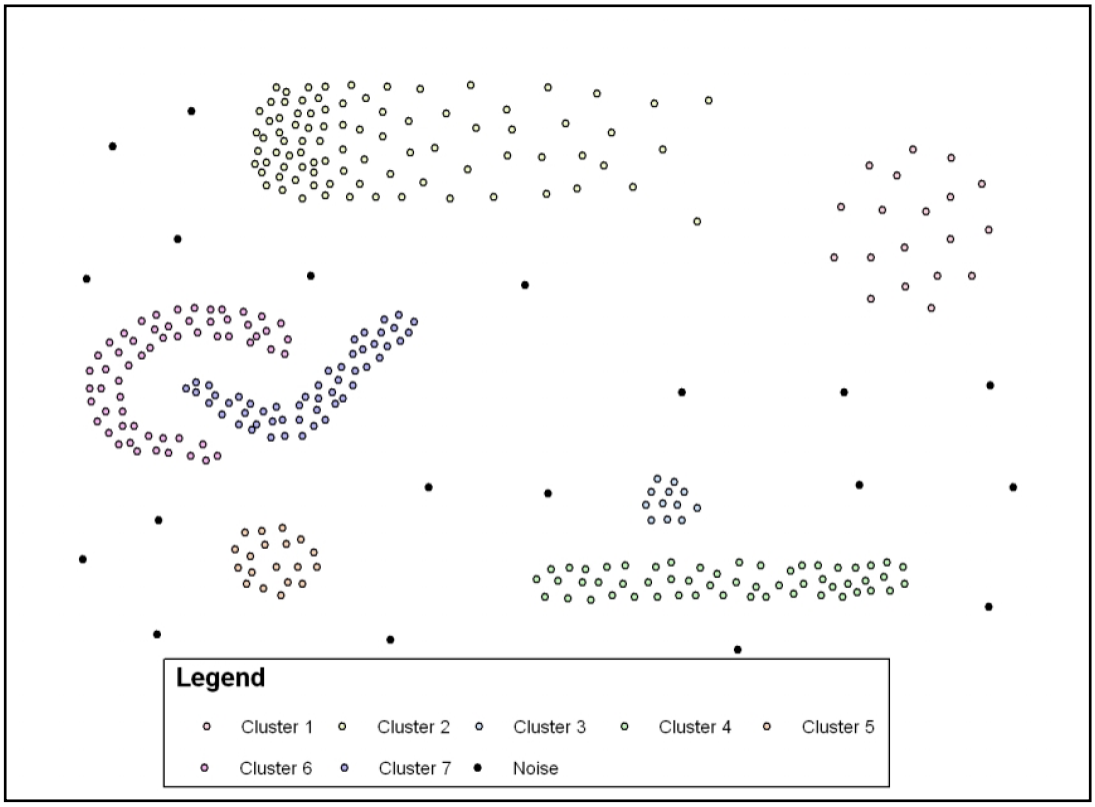

Figure 25: SNN cluster

We reason that the density cluster method an indication is on the type of environment. Setting the parameters according to the average 'cost', we deduce that if all points belong to a cluster, then we work with a clustered environment. If the percentage of noise is not too big, it is probably a random clustered environment. If we reduce the minimum points required, we hope to identify chains that are used in the stop neighbour list.

4.4 Construction Heuristic

The initial solution builds a first round set of routes that is normally used in the improvement stage. Initial solutions tend to follow a greedy algorithm approach which enables them to reach a feasible solution. The execution time is a fraction of the solution time.

The initial solution influences the improvement stage drastically. It is shown that good initial solution can assist in achieving a quicker convergence. We identify three goals for our initial solution:

- High quality – this does not necessarily mean the cost of the solution should be low. The focus is on the initial solution's result that is taken into the improvement stage. The better the combinations, the faster the improvement.
- Initialize parameters – the improvement phase is sensitive to parameters required. The initial solution must assist in accurately set the required decision parameters depending on the information gained.
- Setup memory structures – the information gained during the initial solution stage contains valuable hints that must be utilised by the improvement phase.

The initial solution can be aligned by the developer with the environment by adjusting selection criteria from information available in the problem domain. This thesis is not build on a known environment or for a specific problem type.

Marius Solomon was one of the first researchers to consider the VRPTW. He designed and analysed a number of algorithms to find initial feasible solutions for the VRPTW (Solomon, 1987). His sequential insertion heuristic (SIH) gave very good results in most environments, and most current heuristic methods make use of this heuristic (or a variation thereof) to effectively find a feasible starting solution.

Each customer i has a known demand q_i to be serviced (either for pickup or delivery) at time b_i chosen by the carrier. Because time windows are hard, b_i is chosen within a time window, starting at the earliest time e_i and ending at the latest time l_i that customer i permits the start of service. A vehicle arriving too early at customer j, has to wait until e_j. If t_{ij} represents the direct travel time from customers i to customer j, and s_i the service time add customer i, then the moment at which service begins at customer j, b_j, equals $\max\{e_j, b_i + s_i + t_{ij}\}$ and the waiting time w_i is equal to $\max\{0, e_j - (b_i + s_i + t_{ij})\}$.

After initialising the route, the insertion criterion $c_1\ (i, u, j)$ determines the cheapest insertion place for all remaining, un-routed customers between two adjacent customers i and j in the current partial route $(i_0, i_1, \ldots, i_m)$. Each route is assumed to start and end at the depot $i_0 = i_m$. The indices $p = 1, \ldots, m$ are used to denote a customer's position in the route. The insertion cost is a weighted average of the additional distance and time needed to insert the customer in the route.

Inserting customer u between i and j increases the length of the route by the distance insertion, $d_{iu} + d_{uj} - md_{ij}$. After inserting a customer u between the adjacent customers i and j, a push forward can be calculated for each consecutive node k,

$$PF_k = b_k^{new} - b_k$$

in which b_k (b_k^{new}) denotes the beginning of service at customer k in the route before (after) inserting customer u. The value of PF_k is maximal for the direct successor $k = j$ of u. The sequential insertion heuristic uses the maximal push forward to measure the time needed to insert customer u in the route, the so called time insertion.

The next step of the sequential insertion heuristic decides on which customer to insert the route. The selection criterion c_2 (i, u, j) selects the customer for which the cost difference between insertion in the current or a new route is the largest. This customer is inserted in its cheapest insertion position in the current route. If all remaining un-routed customers have no feasible insertion positions, a new route is initialised and identified as the current route.

We extend the Solomon criteria by utilising the neighbour stop information in testing for a suitable stop to add to the route. Using only stops that have a feasible probability value reduce the number insertion positions to test for each stop. When testing for the insertion position in the current route fails because of the probability, inserting customer u between adjacent nodes for the rest of the route will fail as well. This method will increase the speed of the construction heuristic without diminish the quality of the result.

The algorithm is extended in a bi-directional manner. The criterion c_1 (i, u, j) is extended to criterion c_1 (k, u, i) which represents the insertion of a possible neighbour before stop i. The set from which u is selected is based on the probability order and improvement direction of the result, $u \in N(x_i)$. The algorithm tests the best possible candidates first and then monitors the effect of the subsequent candidates that is ordered from best to worst. If no improvement was achieved in a certain number of iterations, the algorithm terminates the cycle of testing c_1 and continues to the next step.

We also extend the criteria by a Push Backward if a customer is inserted between the depot and the first customer as proposed by Dullaert and Bräysy (2003). If customer u is inserted

between the depot $i_0 = i$ and the first customer $i_1 = j$, a push backward is introduced in the schedule.

Since all vehicles are assumed to leave the depot at the earliest possible time e_i, and travelling from i to j takes t_{ij} units of time, a waiting time of $\max\{0, e_j - t_{ij}\}$ is generated at $j = i_1$. Unlike the waiting time at all other customers $i_r, p < r \leq m$ in the route, it is fictitious. After finishing the route, it can be eliminated by adjusting the depot departure time.

High waiting times stored at customers that used to be scheduled at the first position during the solution construction, cannot be removed this easily. By assuming all vehicles leave the depot at e_0 and by equalling the time insertion to the maximum push forward, the time needed to insert a customer before $i_1 = j$ can be underestimated. It may even be wrongly equalled to zero.

```
Select seed node, most expensive from depot
While n < 5
      For each stop i in route
            Select neighbour n of stop i on the list
            Insert after stop i
            If cost > bestcost then set bestcost
      end for
      n++
end while
if calculated cost of new stop on own route > delta cost
      insert stop on route
```

Algorithm 6: Adapted PFSIH

Algorithm 6 highlights the internal workings of the adapted push forward sequential insertion heuristic. The general technique for implementing a SIH algorithm selects a non-routed stop and tests the stop on each position on the route. This adaption visits each position in the route, but alters the stop being tested to be inserted. The sequence of stops being tested after a specific route stop is determined from an ordered list of non-visited neighbours.

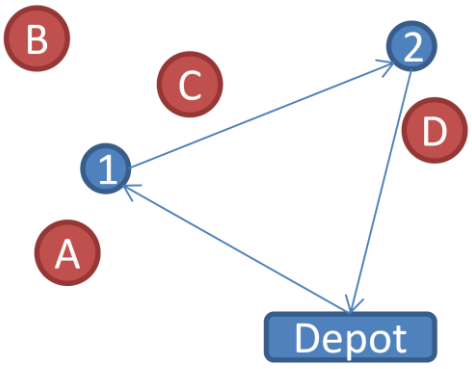

Figure 26: Adapted PFSIH

From Figure 26 we can construct the following table:

Stop	Neigh 1	Neigh 2	Neigh 3	Neigh 4
1	A	C	B	D
2	D	C	B	A
Depot	A	D	C	B

Table 4: Adapted PFSIH Example

If we assume stop 1 and 2 has already been added to the route, the algorithm will test (1,A), (2,D), (Depot,A), (1,C), (2,C), (Depot, D) etc. until the best level found is less than the current threshold minus a constant c. If c = 3 in this scenario, we can see that the algorithm will not test neighbours in column 4.

Although this method is used to setup an initial solution, it can also be adjusted to depend on the probability matrix that was influenced by other operations. The initial solution is build after the probability matrix has been constructed and the environment analysis has been done. The ordered neighbour list that is used for testing can be done on the probability and not the cost.

This greedy approach that also relies on some environmental info provides us with an initial solution that is aligned for improvement as well as indicative of trends. The simplicity of the algorithm ensures a fast execution time.

4.5 Tabu Search

The Tabu Search method is used as a local search technique. A distinguishing feature of Tabu search is its exploitation of adaptive forms of memory, which equips it to penetrate complexities that often confound alternative approaches. The rich potential of adaptive memory strategies is only beginning to be tapped, and the discoveries that lie ahead promise to be as important and exciting as those made to date. Principles that have emerged from the Tabu Search framework give a foundation to create practical systems whose capabilities markedly exceed those available earlier. Conspicuous features of Tabu search are its dynamic growth and evolving character, which are benefiting from important contributions by many researchers.

Tabu search provides a range of strategic options, involving various levels of short term and long-term memory. Consequently, it can be implemented in corresponding levels ranging from the simpler to the more advanced. Generally, the more advanced versions exhibit the greatest problem solving power, though simple ones often afford good results as well. The convenience of building additional levels in a modular design, allowing a Tabu Search procedure to be evolved from the "ground up," is a feature that also provides a way to see and understand the relevant contributions of different memory based strategies.

Implementing a specific strategy for the specified problem is complicated by the fact that we cannot or should not rely on the manner of the problem. As mentioned in the introduction, input data can vary from long haul to short haul, long time windows or short multiple time windows, heterogeneous fleet of similar fleet. To solve the VRP with all its side constraints and unpredictable input data, we implement new operations and add some statistical selection method in the guidance algorithm.

4.5.1 Move Operators

Some meta-heuristics maintain at any instant a single current state, and replace that state by a new one. This basic step is sometimes called a state transition or move. The move is uphill or downhill depending on whether the objective function value increases or decreases. The new state may be constructed from scratch by a user-given generator procedure. Alternatively, the new state be derived from the current state by a user-given mutator procedure; in this case the new state is called a neighbour of the current one. Generators and mutators are often

probabilistic procedures. The set of new states that can be produced by the mutator is the neighbourhood of the current state.

4.5.1.1 Insert Operator

The insert operator tries to insert an orphan stop into an existing route. The method loops through the orphan list of the current solution and calculates a best insertion position. The orphan stop's neighbours are tested for insertion cost. This is done by selecting a neighbour, determining the route the neighbour belongs to and calculates the cost of inserting the orphan stop after the neighbour. If the neighbour is an orphan itself, the test is not done. The method locates a set of closest geographic neighbours from the stop and tests the validity of the insertion of the orphan stop after the neighbour stop. The move is accepted if the insertion is valid.

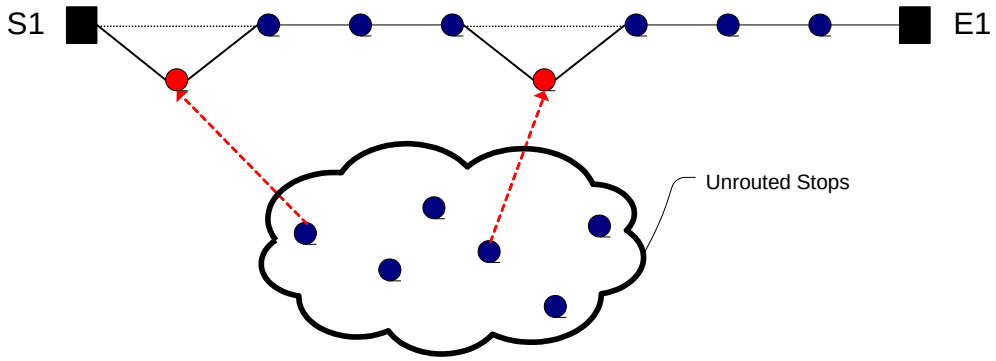

Figure 27: Insert Operation

4.5.1.2 Tour depletion operator

The purpose of this move is to reduce the number of vehicles required to serve all the stops. If it is possible to remove a vehicle, the probability that total distance will decrease is high. It might not be the result in some situations, but the heuristic also depends on diversification.

The procedure looks for the vehicle that contains the least number of stops allocated to routes for the vehicle and is not Tabu. We qualify the routes of a vehicle for removal if the number of stops is less than a percentage of the average number of stops in all the vehicle routes. This is done on the assumption that stops and vehicles have similar characteristics. The difference between stops in terms of volume is assumed to be in a reasonable tolerance.

The first step is to select a tour for depletion according to the criteria specified.

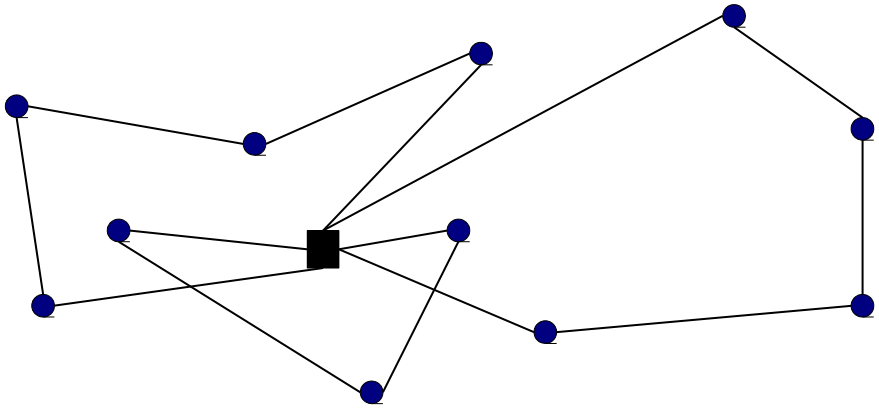

Figure 28: Tour Depletion Step 1

The tour is removed from the solution and the stops belonging to the tour is added to the orphan list.

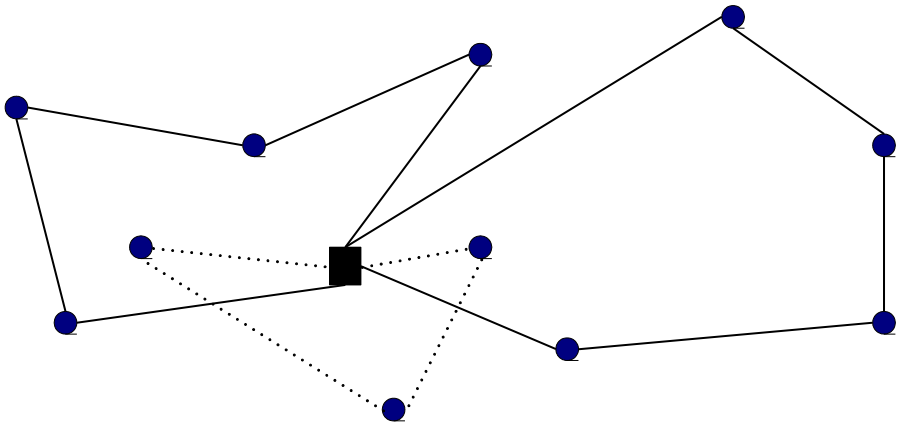

Figure 29: Tour Depletion Step 2

The insert operator is executed to insert the newly created orphans into existing routes.

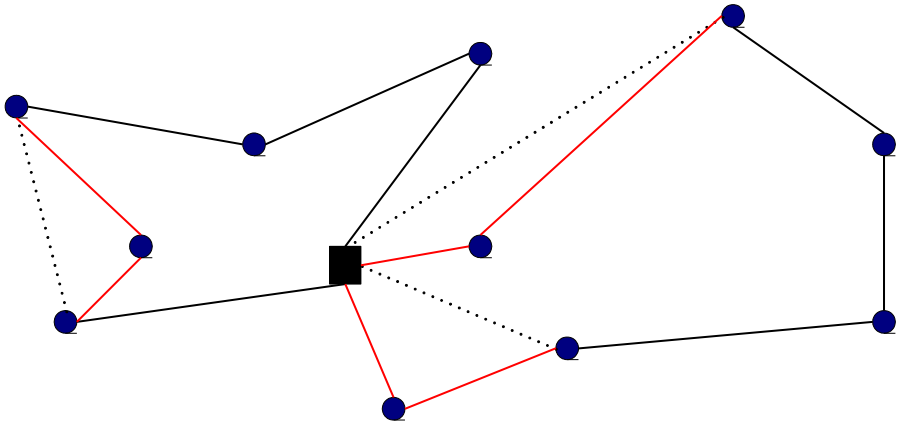

Figure 30: Tour Depletion Step 3

An additional criterion for the tour depletion operator to execute is the non-existence of orphans in the solution. We implement the logic before we even start with actions on the operator, as we assume that if an orphan exists, the current solution is already in such a state that the current route vehicles cannot service all the stops. The meta-heuristic guidance algorithm must execute other operations to optimise the solution that tour depletion is possible.

4.5.1.3 Relocate operator

The relocate operator (Or-opt) removes one stop from a route and inserts it into another route. The implementation group routes to a vehicle and therefore we randomly select a vehicle to add a stop to. Next we randomly select one of the vehicle routes. For each stop on the current vehicle route, an attempt is made to insert a neighbour of the current stop on the current vehicle route. The neighbour is relocated from its route to the current route.

The relocate operator can relocate a stop from the same route to another position.

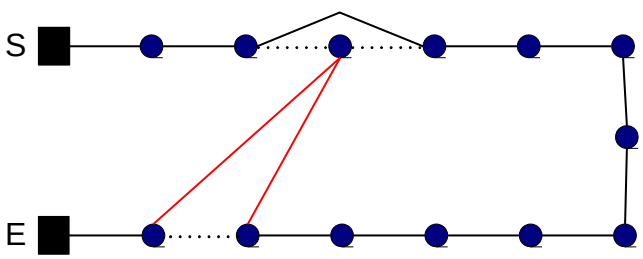

Figure 31: Relocate on same route

Or relocate a stop from one route to another.

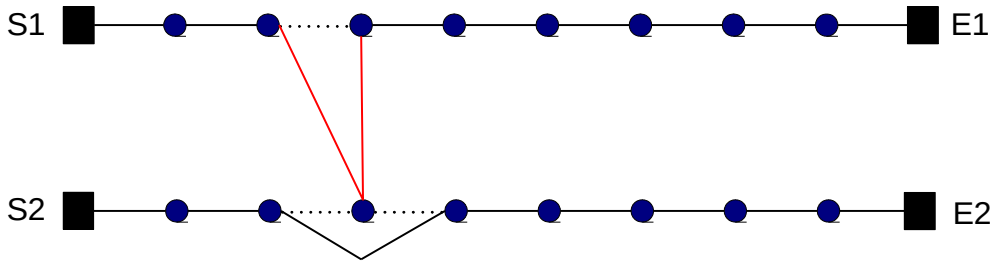

Figure 32: Relocate between routes

4.5.1.4 Exchange Operator

The exchange operator randomly selects a vehicle and corresponding route. The neighbours of the selected route's stops are tested for exchange between the corresponding routes. The operator acts on single stops from different or same routes only.

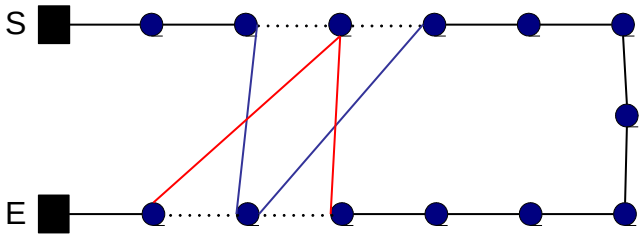

Figure 33: Exchange on single route

The exchange from one route to another simulates a relocate from the one route to the other and vice versa.

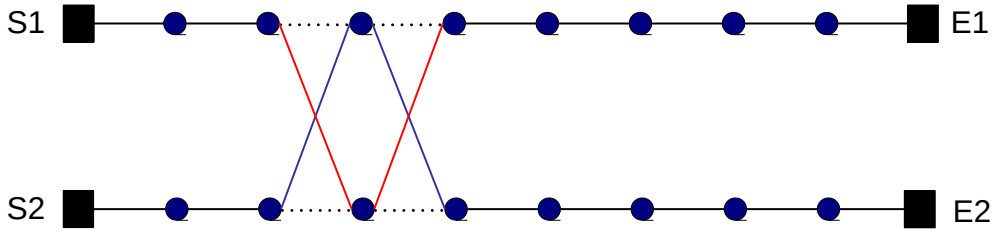

Figure 34: Exchange between routes

4.5.1.5 Cross operator

This operator cuts two routes at a position and swaps the second part of the routes. This is done by selecting a source vehicle and a source route randomly. Each stop in the source route is tested for the move. The stop's neighbours are tested for validity by checking if the stop is not on the same route. If not, the source route consisting of the stops up to the selected stop is combined with the target route consisting of the stops from the neighbour stop to the end to form a new route. The second new route consist of the target route from the beginning to the stop before the neighbour stop and the source route from the stops after the selected stop to the end. If the swap is valid in the current Tabu environment, it will be accepted.

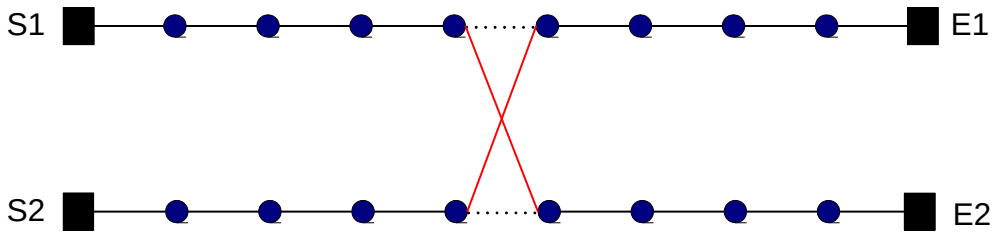

Figure 35: Cross operation

4.5.1.6 Vehicle Fit

This operator exchange vehicles on routes. The operation is added to handle the heterogeneous fleet optimization problem. A vehicle can be swapped between routes if the capacity and time windows allow for the routes qualify.

If there exists vehicles that have not been used, the vehicles can be tested on existing routes to result in better optimization. Tour depletion can result in a more effective vehicle to become available, and the vehicle fit operator will reinsert an available vehicle in the solution.

4.5.1.7 Operator probability

The standard Tabu heuristic is extended with a meta control system on the selected operations. On start of the algorithm, each operator is assigned a weight. All operators start out as equal in the Tabu only solution. In this study, it can differ because of the knowledge gained from the pre-analysis as well as construction heuristic result.

The specific move's probability increases by a constant factor after each successful iteration. The probability is set not to exceed a specific upper bound and because the initial probability

is never reduced, a specific operator will always have likelihood to execute. This memory structure adds an additional dimension to the control of the algorithm. It reacts purely on the output from the move and the decision for the increase is localised to the operation itself. The controlling algorithm which has the ability to select the operation for the next move can use the probability distribution as indicator.

4.6 Simulated Annealing

Simulated Annealing searches the solution space by simulating the annealing process in metallurgy (Qili, 1999). The algorithm jumps to distant location in the search space initially. The size of the jumps reduces as time goes on or as the temperature "cools" down. Eventually the process will turn into local search descent method.

One of its characteristics is that for very high temperatures, each state has almost equal change to be the current state. At low temperatures only states with low energy have a high probability of being the current state. These probabilities are derived for a never ending executing of the metropolis loop.

In the modified version of SA, the algorithm starts with a relatively good solution resulting from a construction heuristic. Initial temperature is set at $T_S = 100$, and is slowly decreased by

$$T_k = (T_{k-1})/(1+\tau\sqrt{T_{k-1}})$$

Equation 3: Cool down tempo

Where T_k is the current temperature at iteration k and t is a small time constant. The square root of T_k is introduced in the denominator to speed up the cool process. Here we use a simple monotonously decreasing function to replace the $1/\log k$ scheme. It is found that the scheme, gives fairly good results in much less time. The algorithm attempts solutions in the neighbourhood of the current solution randomly or systematically and calculates the probability of moving to those solutions according to:

$$P(accepting\ a\ move) = e^{(-\Delta/T_k)}$$

Equation 4: Solution acceptance probability

This is a modified version of the annealing equation, where Δ = C'(S) - C(S), C(S) is the cost of the current solution and C'(S) is the cost of the new solution. If $\Delta < 0$ the move is always warranted. One can see that as the temperature cools, the probability of accepting a non-cost-saving move is getting exponentially smaller. When the temperature has gone to the final temperature T = 0.001 or there is no more feasible moves in the neighbourhood, we reset the temperature to

$$T_r = \max(T_s / 2, T_b)$$

Equation 5: Reset temperature

where T_r is the reset temperature, and was originally set to T_s, and T_b is the temperature at which the best current solution was found. Final temperature is not set at zero because as temperature decreases to infinitesimally close to zero, there is virtually zero probability of accepting a non-improving move. Thus a final temperature not equal but close to zero is more realistic.

4.7 Ant Algorithms

Observations on real ants searching for food were the inspiration to imitate the behaviour of ant colonies for solving combinatorial optimization problems. Real ants are able to communicate information concerning food sources via an aromatic essence, called pheromone. They mark the path they walk on by laying down pheromone in a quantity that depends on the length (cost) of the path and the quality of the discovered food source. Other ants can observe the pheromone trail and are attracted to follow it. Thus the path will be marked again and will therefore attract more ants. The pheromone trail on paths leading to rich food sources close to the nest will be more frequented and will therefore grow faster.

The described behaviour of real ant colonies can be used to solve combinatorial optimization problems by simulation: artificial ants searching the solution space simulate real ants searching there environment, the objective values correspond to the quality of the food sources and an adaptive memory corresponds to the pheromone trails. In addition, the artificial are equipped with a local heuristic function to guide their search through the set of feasible solutions.

It is useful to list some broad behavioural categories which might be classified as collective intelligence, or swarm intelligence (Millonas, 1992). These may be thought of as evolutionary principles of selection, and are not intended to be definitive.

- The first is the proximity principle. The group should be able to do elementary space and time computations. Since space and time translate into energy expenditure.
- Second is the quality principle. The group should be able to respond not only to time and space considerations, but to quality factors, for instance, to the quality of foodstuffs or safety of location.
- Third is the principle of diverse response. The group should not allocate all of its resource along excessively narrow lines. It should seek to distribute its resources along many modes as insurance against the sudden change in any one of them due to environmental fluctuations.
- Fourth is the principle of stability. The group should not shift its behaviour from one mode to another upon every fluctuation of the environment, since such changes take energy, and may not produce a worthwhile return for the investment.

In this thesis we utilize the *concept* of the ant system to solve the VRP. In the traditional way to solve the VRP, the artificial ants construct vehicle routes by successively choosing cities to visit, until each city has been visited. Whenever the choice of another city would lead to an infeasible solution for reasons such as vehicle capacity or total route length, the depot is chosen and a new tour is started. For the selection of a (not yet visited) city, two aspects are taken into account: how good was the choice of that city, information that is stored in the pheromone trails $\tau_{i,j}$ associated with each arc (v_i, v_j), and how promising is the choice of that city. This latter measure of desirability, called visibility and denoted by η_{ij}, is influenced by factors calculated from previously discussed tools. In the case of the VRP on Solomon's benchmark, the desirability is defined as the reciprocal of the distance, i.e. $\eta_{ij} = 1/d_{ij}$.

With $\Omega = \{v_j \in V : v_j \; is \; feasible \; to \; be \; visited\} \cup \{v_0\}$, stop v_j is selected to be visited after stop v_i according to a random-proportional rule in that can be stated as follows:

$$p_{ij} = \left\{ \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{h \in \Omega}[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}} \right\} \text{if } v_j \in \Omega, 0 \text{ otherwise}$$

Equation 6: Random-proportional rule

This probability distribution is biased by the parameters α and β that determine the relative influence of the trails and the visibility, respectively.

After an artificial ant *k* has constructed the feasible solution, the pheromone trails are laid depending on the objective value L_k. For each arc (v_i, v_j) that was used by the ant *k*, the pheromone trail is increased by $\Delta\tau_{ij}^{k} = 1/L_k$. In addition to that, all arcs belong to the so far best solution (objective value L^*) are emphasised as if σ ants, so called elitist ants had used them. One elitist ant increases the trail intensity by an amount $\Delta\tau_{ij}^{*}$ that is equal to $1/L^*$ if arc (v_i, v_j) belongs to the so far best solution, and zero otherwise. Furthermore, part of the existing pheromone trails evaporates (ρ is the trail persistence). Thus, the trail intensity is on update according to the following formula, where *m* is the number of artificial ants:

$$\tau_{ij}^{new} = \rho\tau_{ij}^{old} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + \sigma\Delta\tau_{ij}^{*}$$

Equation 7: Pheromone intensity update

The constructive method of building the routes, forces the initial placement of ants at each stop. The implication for the implementation of this Ant System on the VRP is that as many ants are used as there are customers in the VRP, and that one ant is placed at each customer at the beginning of iteration. After initialising the basic ant system algorithm, the two steps construction of vehicle routes and trail update, are repeated for a given number of iterations. This solution might be sufficient for the travelling salesman problem, but is no where efficient enough for a complex VRP problem.

This constructive methodology does not support an efficient heuristic approach. The side constraints of the VRP contribute to the complexity of valid stop selection in the latter part of the route. The combinations of selected stops do not necessarily result in a good solution because of the structure of the environment. In a low constraint impact scenario, i.e. where a

stop has a number of possible neighbours, the constructive method can result in good and even good enough solutions, because of the possible combinations still left at the latter part of the route. On the other end of the spectrum, where constraints are strict, the method can also result in good solutions because of the limited number of combinations that exist in any case across the entire solution.

The methodology of using ants to build routes has to be extended to allow for a feasible methodology. Let us consider the analogous between the first research of termites by Eugene Marais and the level of implementation achieved. While observing the natural behaviour of these creatures, he noticed that firstly, the whole termitarium had to be considered as a single organism whose organs work like those of a human being. The queen was the brain and the womb; the workers were mouthparts and tissue builders; the soldiers were the white blood cells and the humus gardens were the stomach. Then secondly, he noticed that the actions within the termitarium were completely instinctive.

Compared to this, the current ant approach is just half a brain and mouthparts. This study improves on the implementation approach, and not just on the formulas used to calculate the pheromone trial, which determine the selection criteria. The consideration of the problem environment, as well as the creative construction methods build up the new body.

It is important to realise that the real world ant problem has the major advantage to ignore poor trials. This can be translated in the VRP to the generation of orphans when the stops are just too costly. The idea seems lucrative in defining business on a strategic level. This study does not allow orphans as part of the solution. The design of such a solution has an impact on the objective function that can be handled by this solution approach, but the impact on the search algorithm is not considered in this study. It can be investigated in further studies.

Studies of the foraging behaviour of several species of real ants revealed an initial random or chaotic activity pattern in the search for food. Traditional two stage approach in the VRP suggests that the initial solution should be as good as possible. In this study we revert back to the scientists' suggestion and emphasised the importance of the initial preparation. A 'good' initial solution is redefined from the traditional value based evaluation. The convergence to a good solution is dependent on the common iteration between the various ants. The recruitment mechanism differs for different species, although the most common use is the pheromone trial.

Hybridisation in general means combining ideas of several different methods in one approach. Such proceeding is common practice for hard combinatorial optimisation problems and has been successfully applied to other problems. We define the hybridisation as different functions by different body parts in the ant system. The environment influences the brain to use certain parts of the body more than other, or in combination with other.

Mapping our solution to the human body analogy of Marais, the following ants with their associated memory structures are defined in the system:

- Queen – The queen is the brains of the operation. This ant is represented by the Ant System Class and is responsible for controlling all actions.
    - Simulated Annealing – the queen utilise the adapted SA approach as described in a previous paragraph to guide the goals of the scouts and workers.
    - Probability – the queen control the overall probability matrix gained from all ants. The probability value is based on an extended formula on the traditional pheromone calculation.
    - Environment – the queen deduces the environment from statistical data and scout ants.
    - Parallel – the queen controls the parallel processing of all ants.
- Scouts – Scouts are defined as ants that do a random chaotic search, to ensure that divergence is achieved. Scouts do have a certain level of intelligence and can control subordinates from their knowledge gained.
    - Initial solution – the scout's main function is to construct an initial solution base to work form.
    - Clusters – the scouts use the clusters as indication of areas of similarity.
    - Inverse pheromone – the scouts use the provided pheromone trial as tabu area to ensure convergence.
- Soldiers – Soldiers are responsible for the optimisation of the current operations. Soldiers are also equipped with some specific level of intelligence which is used to guide the workers.

- Local heuristic – the soldiers use the Tabu search heuristic to improve on the solutions.

- Workers – Workers are responsible to build the empirical proof of the solution. The aim is that they do not have intelligence over a group, but is focussed on their immediate environment.
  - Pheromone trial – the workers build routes through a traditional pheromone trial.
  - Memory – the workers react on memory set by themselves as well as fed down from their controlling ant.

The subsequent diagram in Figure 36 depicts the relation between the entities. It can be viewed as a hierarchal system.

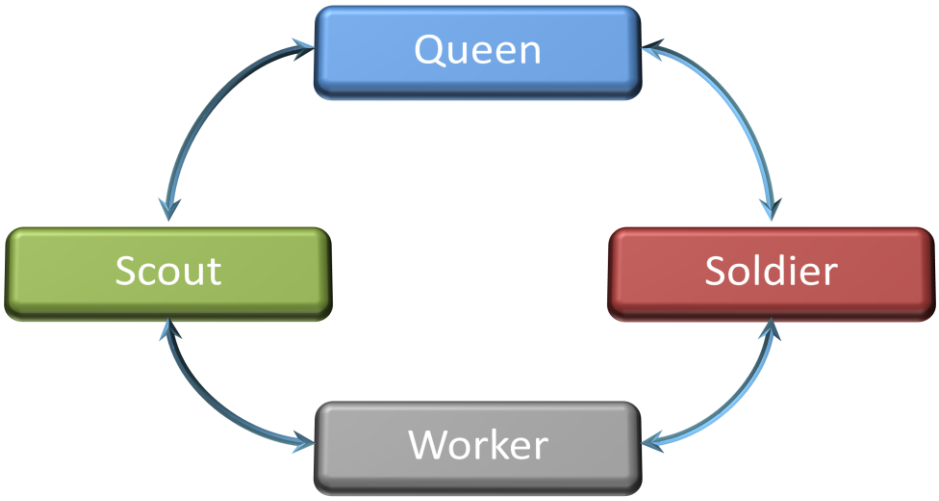

Figure 36: Ant type relations

The aim of the ant solution is the principle of locality. The behaviour of the individual organisms will be determined solely by local influences. This means that the individual organisms will not have any memory, non-local navigational skills, or any type of behaviour that involves storage of internal information. Any information flow must then be a product of the collective behaviour. The communication between the different types of ants is to influence the local stored information.

4.8 Solution Algorithm

This section summarise the algorithm designed in this study. Literature has shown that methods like Tabu are still superior to new agent-based approaches. Results will show that the intelligent implementation of agents and controlling the relations between them can both utilise proven local methods, but also increase the time to find a good solution in real-world applications. The Solomon benchmark problems are used as a base, and several other problem spaces are solved to show the flexibility of the algorithm.

The difference between an agent based approach and typical Tabu searches is not that clear cut. We argue that the agent based approach is a conceptual structure that assists with the management of the typical adaptive memory. The agent based approach also add another dimension, the inter agent communication layer. This approach assists in the component orientated design approach and compliments the use of adaptive objects.

The principle of locality is tricky to implement. The approach is to ensure that an intelligent ant is well defined, i.e. it is familiar with all parameters available to use and how to react. The inter ant communication can influence the local parameters. We reason that a too simplistic implementation of a pheromone trial will not suffice as a practical approach to our problem. Creating a more complex structure is necessary, but it can be done on a different layer to ensure the problem complexity is still manageable.

```
Read problem space from adaptive provider.
Create solution space from problem space
      Calculate a best possible cost matrix
      Deduct a probability matrix from cost matrix
Evaluate environment
      Count number of possible neighbours per stop
      Calculate average, best and worst cost per stop
      Compare values with neighbour and overall stops
Apply environment result on probability matrix
      Decrease probability on unlikely stops
Solve
      Initial solution
      Improvement solution
```

Algorithm 7: Solution Approach

Algorithm 7: Solution Approach defines the outline of our Ant System on Adaptive Objects (ASAO) algorithm. It is important to keep in mind that the algorithm is build on top of an adaptive object model. It has to provide the guideline to the problem space elements on how it could be used optimally, i.e. when are the function calls to the object cost functions.

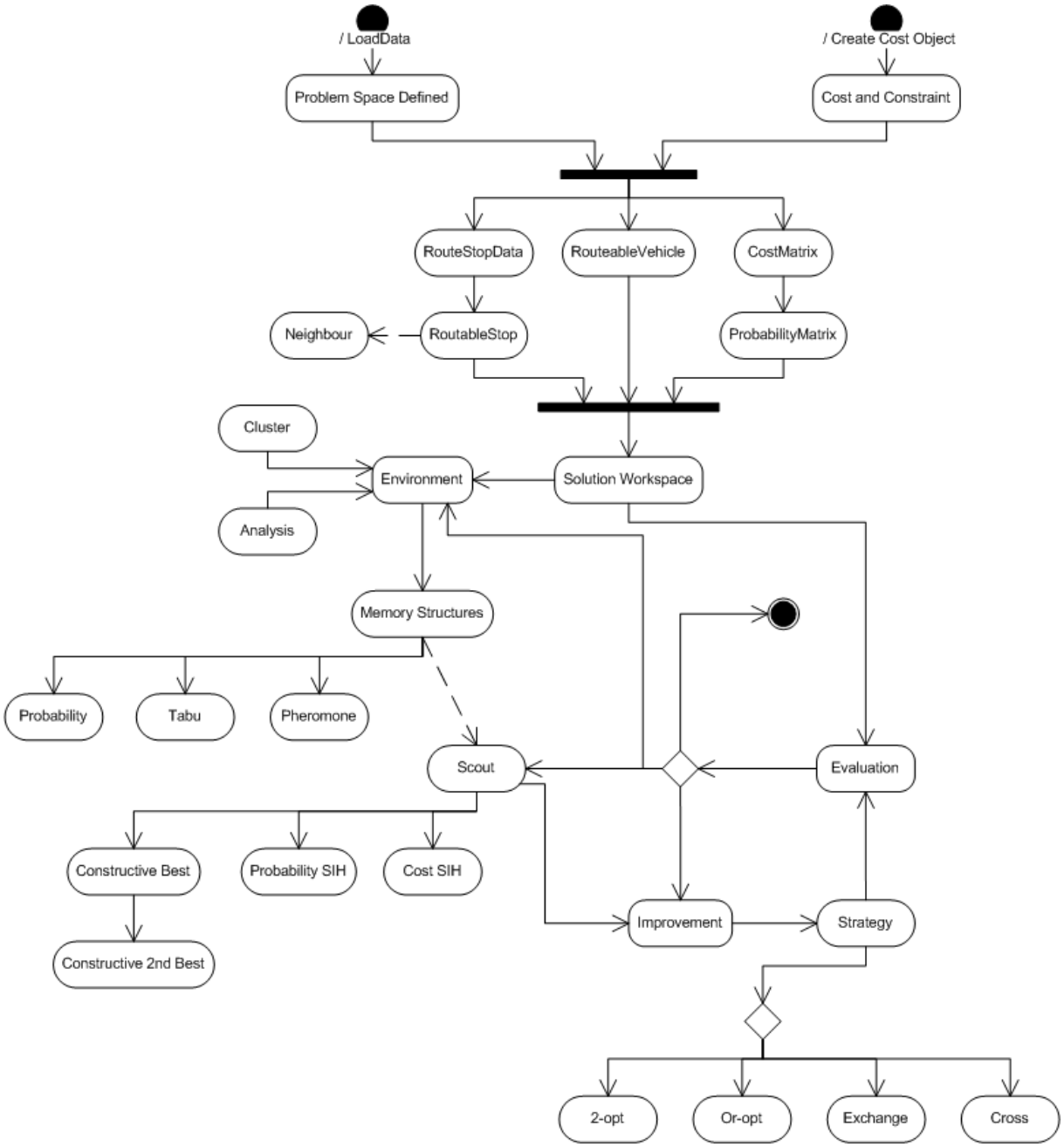

Figure 37: Ant System on Adaptive Objects - ASAO

Algorithm 8 represent a pseudo version on high-level for the ASAO algorithm.

```
Step 1.  Build solution workspace from problem space.
      a. Build best cost matrix
      b. Build initial probability matrix
      c. Create routable stops and vehicles
      d. Build environment summary
Step 2.  Initialise ASAO
Step 3.  Send out scouts
      a. Constructive ant best accept
      b. Constructive ant second best accept
      c. Constructive ant random accept
      d. Constructive ant adapted neighbour insertion
Step 4.  Evaluate results and update memory structures
Step 5.  Apply selective soldiers on selective solutions
Step 6.  Evaluate best solutions, if improvement deemed
   possible, go to 4
Step 7.  If improvement stabilize, go to 3
Step 8.  Update possible final solution list
Step 9.  If number of iterations < max number go to 3.
Step 10. Return final solution list.
```

Algorithm 8: ASAO

The final result consists of a list of possible solutions that represent a dissimilar solution.

4.9 Summary

The solution algorithm depends on the collaboration between several existing techniques. The agent based approached, build on an ant colony optimisation technique, assist in mimicking a multi-agent approach that has strong ties to physical examples which can be empirically proofed.

Solving an already complex problem is an unknown environment requires a hybrid of more than one approach. The multi-start initial solution approach provides a good starting point for the improvement phase. It also influence the memory structure used in the improvement phase, which kick start the improvement phase with knowledge gained.

The combination of an ant pheromone trial and tabu list results in a dual contradictory memory list. The pheromone indicates the better moves, but the tabu control the overall use of these combinations. The implementation of both these lists allows the control mechanism to determine convergence and diversification without explicit events set.

The methods used in this solution consist of well known understood approaches. The powerful use of memory structures to guide the algorithm in an unknown domain is what makes this solution successful.

5 ENVIRONMENT ANALYSIS - RESULTS

5.1 Overview

This chapter discuss examples of the results achieved from specific methods selected in the study. It explains the environments and compares results of different methods for the different environments.

The results are based on the implementation of Solomon's cost and constraint function. This implementation is sufficient to prove other implementations on the adaptive object model. The next paragraph defines our implementation method for the Solomon cost functions.

5.2 Solomon Functions

Solomon generated six sets of problems. Their design highlights several factors that affect the behaviour of routing and scheduling algorithms. They are: geographical data; the number of customers serviced by a vehicle; percent of time-constrained customers; and tightness and positioning of the time windows.

The geographical data are randomly generated in problem sets R1 and R2, clustered in problem sets C1 and C2, and a mix of random and clustered structures in problem sets by RC1 and RC2. Problem sets R1, C1 and RC1 have a short scheduling horizon and allow only a few customers per route (approximately 5 to 10). In contrast, the sets R2, C2 and RC2 have a long scheduling horizon permitting many customers (more than 30) to be serviced by the same vehicle.

The customer coordinates are identical for all problems within one type (i.e., R, C and RC). The problems differ with respect to the width of the time windows. Some have very tight time windows, while others have time windows, which are hardly constraining. In terms of time window density, that is, the percentage of customers with time windows, he created problems with 25, 50, 75 and 100 % time windows.

The larger problems are 100 customer Euclidean problems where travel times equal the corresponding distances. For each such problem, smaller problems have been created by considering only the first 25 or 50 customers.

The implementation requires us to define the domain model as well as the cost and constraint functions for Solomon. Solomon implements a basic VRPTW problem. The following diagram represents the implemented stop class used in the solution.

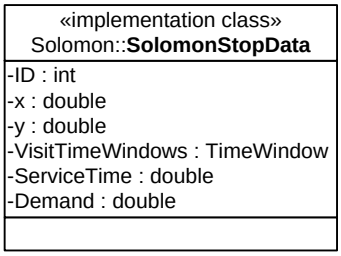

Figure 38: Solomon Domain Instance

Solomon constraints include the following checks:

- Time Windows – each stop has an open and close time window in which the stop must be services. Time windows create a *wait time* that must be considered in the cost.
- Volume – there exist a capacity constraint on each vehicle used in the solution. The volume creates a *service time* at a stop which can be considered at the cost function.

Distance in Solomon does not have a constraint, but to improve cost calculation, distance changes can be updated during a constraint check as constraints are called with a special instruction. Each *routestop* implements an abstract of the *RouteStopData* class which acts as special container for specific implementations. The Solomon implementation can be defined as follows:

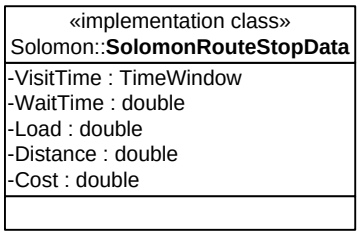

Figure 39: Solomon *RouteStopData* implementation

This adaptive implementation class extends the algorithm memory which contributes to efficiency. The effect of the implementation depends on the implementer's knowledge of the problem domain. The attributes of this Solomon implementation class is as follows:

- Visit Time – the arrival and departure time of the vehicle at the specific route stop for the specified route.
- Wait Time – the time a vehicle is too early and has to wait for the closest forward opening time to occur.
- Load – the total load on the vehicle for the route to the specific stop.
- Distance – the total distance travelled for the route to the specific stop.
- Cost – the total cost incurred for the route to the specific stop.

The efficiency of the adaptive model can now be explored. These structures allow the operator to implement effective methods for calculating cost and checking constraints. The knowledge the operator has of the problem environment can assist in implementing another level of meta heuristic. The basic rules followed in this thesis for Solomon is as follows:

1. Changes to a route on the stop list impacts only attributes of route stops after the stop.
2. Solomon use Euclidean distance and time which comply to the triangular rule:

$$d_{ij} \leq d_{ik} + d_{kj}\,.$$

5.3 Probability matrix

The probability matrix is an important memory structure which guides the algorithm. The analysis of the environment is used to assist in the evaluation of the surface chart shape of possible neighbouring solutions. Constraints such as time windows contribute to the shape of possible solutions. The results display some common patterns that exist in the benchmark problems.

	Depot	Stop 1	Stop 2	Stop 3	Stop 4	Stop 5	Stop 6	Stop 7	Stop 8
# Neighbours from stop	100	3	9	97	15	98	29	75	68
Sum of cost	2885	76	226	3109	368	3106	874	2443	2295
$min\ P(x_{ij})$	11%	88%	83%	13%	78%	12%	67%	31%	36%
$max\ P(x_{ij})$	90%	90%	90%	90%	90%	90%	90%	90%	90%
# Neighbours to stop	100	93	88	15	80	2	76	21	32
Sum of cost	2885	3133	3061	410	2749	20	2622	620	1018
$min\ P(x_{ij})$	54%	40%	38%	42%	40%	70%	39%	42%	40%
$max\ P(x_{ij})$	120%	90%	90%	90%	90%	90%	90%	89%	90%

Table 5: C105 Statistic summary extract

The first column indicates that each stop is reachable from the depot. If this was not true, we were faced with an infeasible solution space. The average cost per stop can be used as benchmark for other stops. The rest of the columns represent the statistics per stop. The extract in Table 5 is only for stop 1 to 8 in Solomon's C105 problem. The summary provides an indication of the influence of constraints on the problem and it can be clearly deducted that constraints effect varies from stop to stop.

Stop 1 has the definite ability to only go to 3 other stops, while 93 other stops can have stop 1 as a subsequent neighbour. The probability distribution between the 3 possible subsequent stops is between 88% and 90%, which can be seen as high enough to know that the likelihood of all of the 3 stops to be the neighbour in the final result. These values are used in the pheromone trial. The statistic can immediately be applied on the initial pheromone trial.

The significant difference between from and to stop statistics indicate on a reduced number of good solutions that exist. It supports the intuition that a good or even best solution can be

reach through a limited number of moves. The statistics does not reveal anything clear on groups of stops, and is used in guiding the improvement stage.

	Depot	Stop 1	Stop 2	Stop 3	Stop 4	Stop 5	Stop 6	Stop 7	Stop 8
# Neighbours from stop	100	31	99	56	34	99	70	86	75
Sum of cost	2495	849	2919	1696	1078	2920	1834	2618	2549
$min\ P(x_{ij})$	11%	65%	12%	46%	63%	12%	35%	22%	31%
$max\ P(x_{ij})$	90%	90%	90%	90%	90%	90%	90%	90%	90%
# Neighbours to stop	100	99	48	84	99	38	77	66	73
Sum of cost	2495	2832	1407	2774	3433	1065	1993	1959	2347
$min\ P(x_{ij})$	52%	45%	37%	34%	30%	36%	42%	38%	31%
$max\ P(x_{ij})$	93%	90%	87%	90%	89%	87%	90%	89%	90%

Table 6: R205 Statistic summary extract

The statistics predicts what we already know from the domain environment of the R205 problem. The number of stops from and stops to as neighbours are much more evenly spaced, which indicates more options of combinations and thus a bit more difficult to deduct an optimal solution from limited moves. The difference between the minimum and maximum probability does however indicate that quality of combination can be used to influence decision making on moves. The pheromone trial can immediately reflect these properties.

	Depot	Stop 1	Stop 2	Stop 3	Stop 4	Stop 5	Stop 6	Stop 7	Stop 8
# Neighbours from stop	75	74	74	74	74	74	74	74	74
Sum of cost	1815	2343	1933	2242	1830	2325	1856	2118	2140
$min\ P(x_{ij})$	11%	12%	12%	12%	12%	12%	12%	12%	12%
$max\ P(x_{ij})$	90%	90%	90%	90%	90%	90%	90%	90%	90%
# Neighbours to stop	75	74	74	74	74	74	74	74	74
Sum of cost	1815	2343	1933	2242	1830	2325	1856	2118	2140
$min\ P(x_{ij})$	53%	31%	43%	37%	49%	31%	49%	38%	37%
$max\ P(x_{ij})$	93%	89%	90%	90%	88%	89%	90%	90%	90%

Table 7: P n76 k4 Statistic summary extract

The statistics in Table 7 indicate a problem space where constraints do not play any role in the preliminary evaluation of the environment. This can be seen from the equal number of possible from and to stops. The extract only show up to stop 8, but all stops showed equal

result for number of possible from and to stops. A stop is the neighbour of every stop in the problem space, except itself. If we look at the domain definition of this problem, it consists of 75 stops and can be optimally solved with 4 vehicles. There is no time constraint on the problem and therefore all combinations of stops are possible. Capacity constraints' impact can only be seen when building a route.

Our first step in solving the problem proved to indicate some trends in the environment. We compared the computed results with the common domain knowledge, and the indication provided by the results is clear enough to use as guidance for the algorithm. We can now add more information through other methods.

5.4 Density cluster results

The following section explains the advantage of density clustering on certain problem environments. Density clustering assists in environments where natural groups exist. In a real-life environment, locations can be grouped in residential areas that are split by commercial properties, nature, roads, etc. Clustering data can assist to generate good solutions quick and effective.

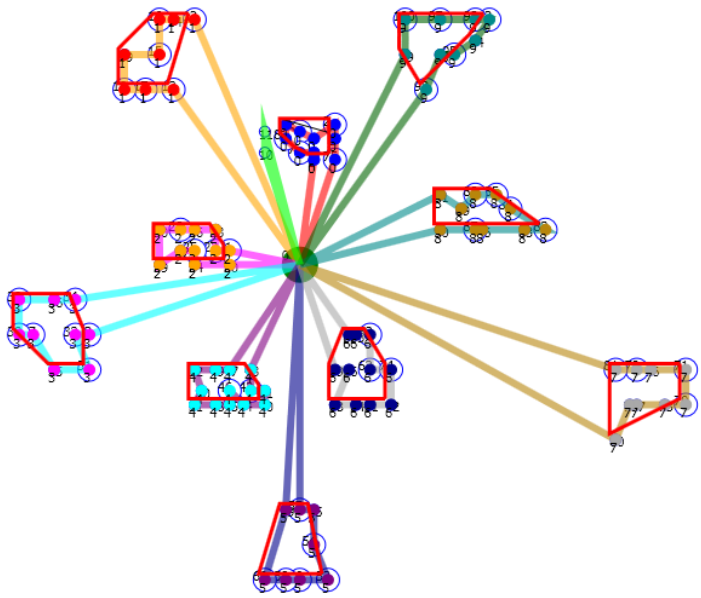

Figure 40: C105 SIH on density cluster

Problem class C1 is specifically designed to benchmark solution algorithms against clustered solutions. The image in Figure 40 is the result of the best initial solution from the algorithm. It is clear that the density clusters forced the initial algorithm into a near optimal solution. The red polygon areas indicate the clusters build from the probability matrix that was constructed in the solution preparation phase. One of the aims for this study is also to provide alternative solutions which might not be theoretical the best, but can indicate to the logistic manager what other possibilities exist. This initial result indicates that alternative solutions that have to be dissimilar will be tough to achieve.

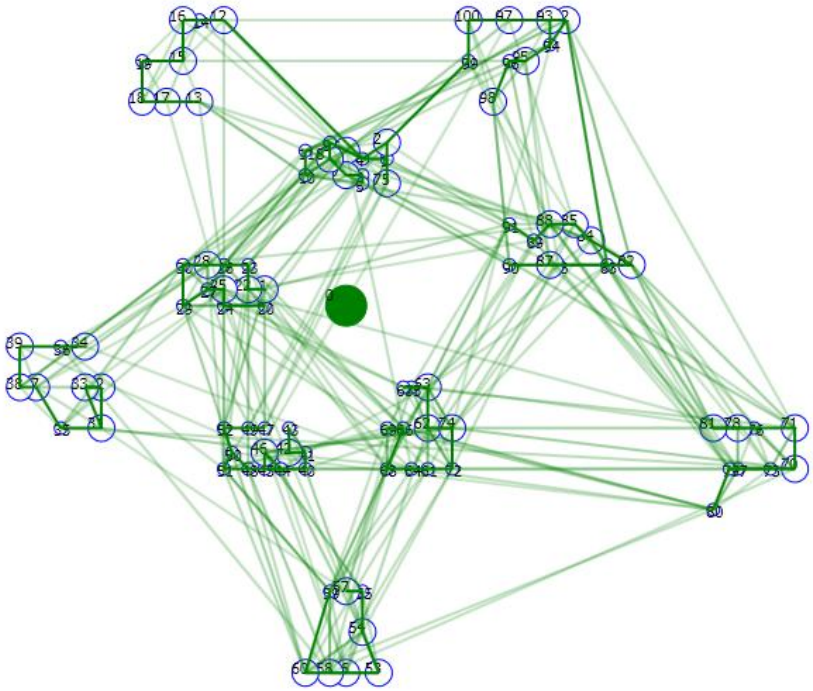

Figure 41: C105 initial pheromone trial

The initial solution algorithms consist of more than just the clustered approach. To ensure that anomalies are also catered for in extreme cases, the pheromone trial is still build up from other results as well. Figure 41 represents the initial pheromone trial before the improvement phase starts. It is clear from comparing with Figure 40 that the clustered stop segments have been given a significant higher probability.

Density clustering is a very effective method for specific cases. Because the method relies on specific criteria to generate a cluster, some environments can result in none or few clusters. Although it still assists in guiding the pheromone trial, we continue to apply additional methods in searching for guidance.

5.5 Partition cluster results

Clusters generated by a partition algorithm do not necessarily reflect a strong binding between the elements as with density clusters. Partition clustering is therefore only used when no or a small number of density clusters exist. The partition cluster method requires a predefined number of clusters to be generated and does not guarantee the same result if the algorithm is executed again.

The use of partition cluster is to accelerate the convergence to a good feasible solution. The number of clusters is not defined as the expected final number of routes, but rather large enough to generate chains to work with.

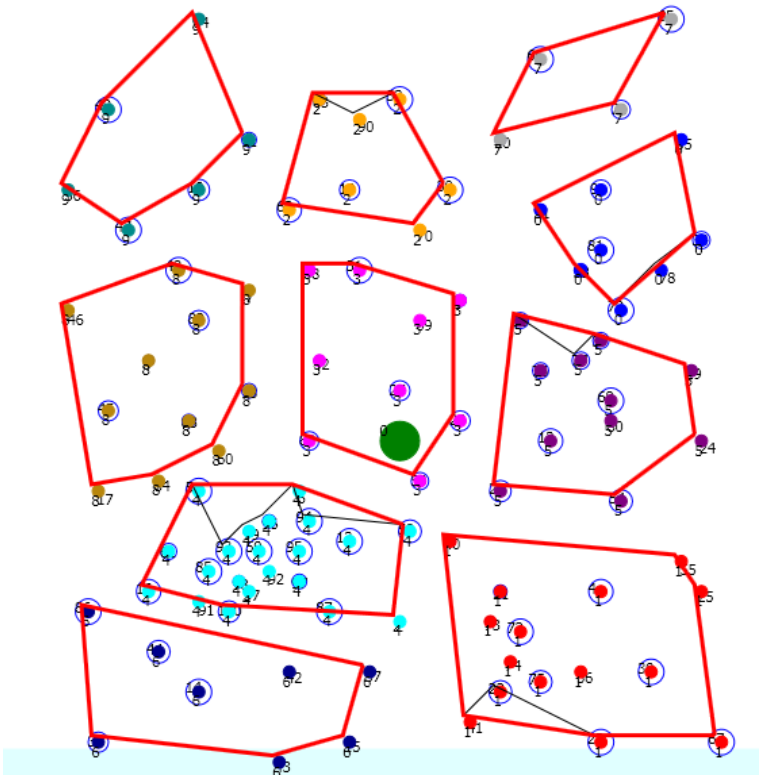

Figure 42: R108 Partition cluster

The R108 Solomon benchmark problem is a random generated environment with a short scheduling horizon. The partition clustering method is non-deterministic algorithm which allows multiple outcomes. Clustering still reflects something from the environment.

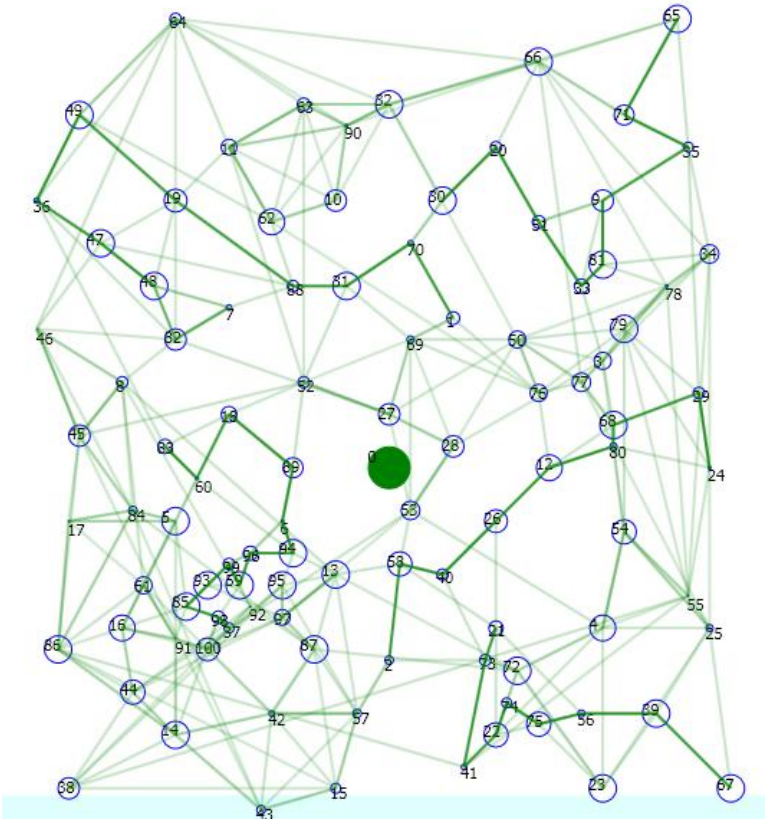

Figure 43: R108 Initial pheromone trial

From the pheromone trial in Figure 43, it is clear that the partition clustering did not affect the initial trial as much as the density method. This is due to the contribution of other factors that did not correlate with all the cluster chains.

5.6 Benchmark Results

This paragraph lists the execution results on some benchmark problems. The main focus is on Solomon's problems. Other examples use the same cost calculation method of Solomon. Executing the algorithm on Solomon's benchmark problems returned very good results.

The best results are from Sintef's website (Solomon-benchmark, 100-customers, 2010) . The tables represent the

- Problem – e.g. C101, RC104, etc.
- Initial Solution – according to a PBIH (Number of vehicles and distance travelled, Moolman (2004))
- Tabu improvement applied (Number of vehicles and distance travelled, Moolman (2004))
- Initial Solution – according this multi-start method (Number of vehicles and distance travelled)
- Improvement applied (Number of vehicles and distance travelled)
- Best published (Number of vehicles, distance travelled and % difference of the study's best) – from Sintef website (Solomon-benchmark, 100-customers, 2010)

5.6.1 C1

The following diagram (Figure 44) display the solution of Solomon's C101 problem. The left diagram is the final routes and the right diagram is the final pheromone trail. The total route distance for the solution is 828.94, which match the best published solution. The clear pheromone trial indicates that alternatives to these routes are not feasible to consider.

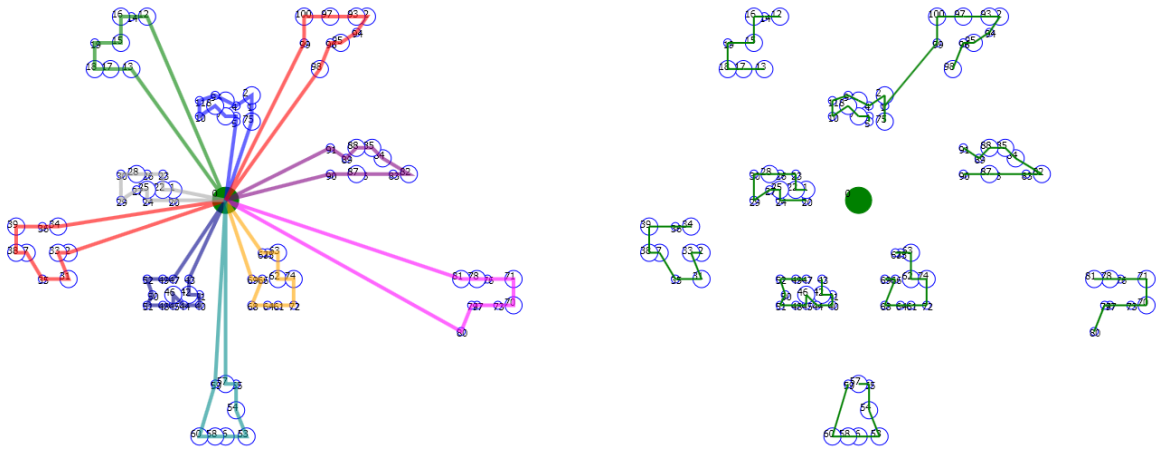

Figure 44: Solomon C101 Solution

The following diagram (Figure 45) display the overlay of the initial clusters overlaid with the solution. The red polygon and same colour stops indicate a cluster. Although the C101 is a

simple solution, it indicates that the cluster should influence the initial pheromone trail as well as Tabu List.

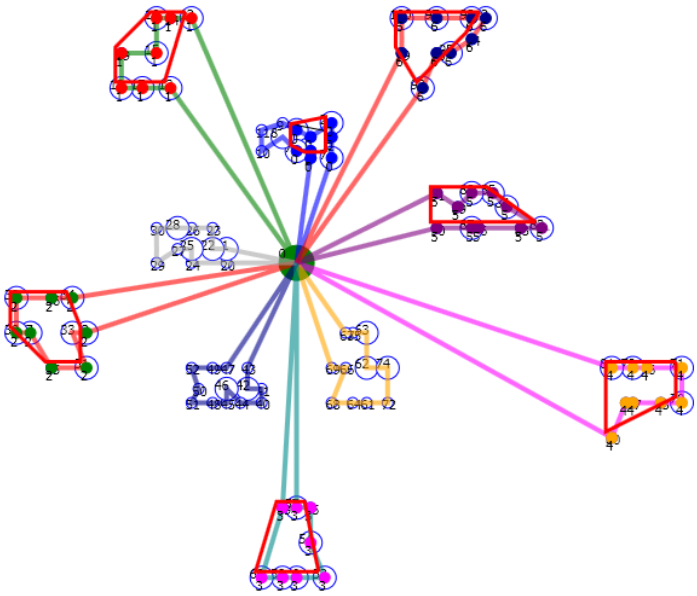

Figure 45: Solomon C101 Cluster overlay

The following diagram (Figure 46) display the solution of Solomon's C109 problem. The total route distance for the solution is 828.94, which match the best published solution.

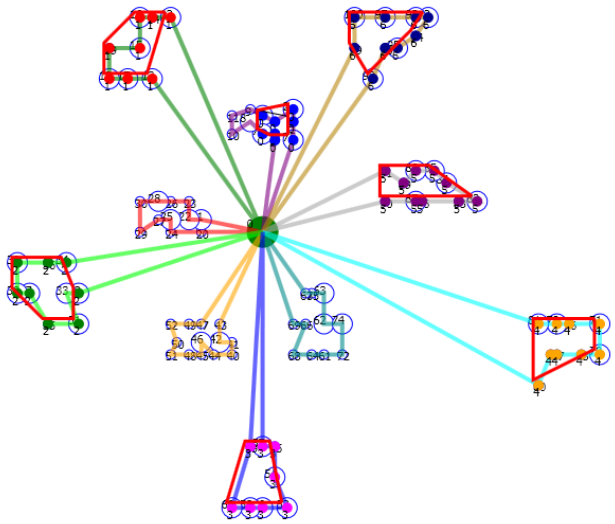

Figure 46: Solomon C109 solution

Table 8 represents the solutions compared to a plain Tabu solution, the best initial solution, the adaptive object algorithm and the best published solution.

			Tabu		New					
Prob	Initial Solution		Improvement		Initial Solution		Improvement		Best Published	
C101	10	880.47	10	828.94	11	860.58	10	828.94	3.7%	10	828.94	**0.0%**
C102	10	997.74	10	871.32	16	1,144.04	10	828.94	27.5%	10	828.94	**0.0%**
C103	10	1081.56	10	916.83	16	1,237.34	10	853.72	31.0%	10	828.06	**3.1%**
C104	10	1059.59	10	911.85	12	975.23	10	836.84	14.2%	10	824.78	**1.5%**
C105	10	878.78	10	827.55	11	860.58	10	828.94	3.7%	10	828.94	**0.0%**
C106	10	968.58	10	840.19	13	976.02	10	834.23	14.5%	10	828.94	**0.6%**
C107	10	928.74	10	827.55	11	860.58	10	834.23	3.1%	10	828.94	**0.6%**
C108	10	871.57	10	827.55	12	910.35	10	828.94	8.9%	10	828.94	**0.0%**
C109	10	910.28	10	829.74	13	979.95	10	850.26	13.2%	10	828.94	**2.6%**

Table 8: C1 Solutions

5.6.2 C2

The setup of the cost functions in the adaptive object model in this contains no cost for the usage of an extra vehicle. This can already be seen by the result of the best initial solution from the multi-start approach. It is assumed that the traditional SIH best solution is contained in the set of initial solutions, but is not reflected as the best. That is evident by comparing the

original initial solution with the new multi-start result. The new algorithm did fairly well on the optimisation for C2 type problems.

	Tabu				New						
Prob	Initial Solution		Improvement		Initial Solution		Improvement			Best Published	
C201	3	826.15	3	588.88	5	673.69	3	591.56	12.2%	3	591.56	0.0%
C202	3	1180.34	3	623.46	6	720.65	4	619.44	14.0%	3	591.56	4.7%
C203	3	1173.25	3	625.46	4	838.94	4	617.20	26.4%	3	591.17	4.4%
C204	3	1235.70	3	685.10	5	870.92	3	646.54	25.8%	3	590.60	9.5%
C205	3	789.79	3	617.45	4	777.30	3	589.72	24.1%	3	588.88	0.1%
C206	3	934.87	3	629.63	5	875.20	3	591.35	32.4%	3	588.49	0.5%
C207	3	884.44	3	587.89	5	869.49	3	601.92	30.8%	3	588.29	2.3%
C208	3	815.97	3	592.93	5	759.59	3	594.73	21.7%	3	588.32	1.1%

Table 9: C2 Solutions

5.6.3 R1

The effect of the randomly distributed stops in the R1 environment result in the inefficient use of the clustering approach for the initial solution. The optimisation produced good results and we can argue that the grouping during the initial solution assisted to guide the meta-heuristic in the correct areas. The greedy nature of the initial solutions resulted in unordered stops, but the heuristic's swap moves improved the result with a low computing cost.

	Tabu				New						
Prob	Initial Solution		Improvement		Initial Solution		Improvement			Best Published	
R101	20	1857.93	20	1,670.13	25	2264.223	21	1673.5645	26.1%	19	1,645.79	1.7%
R102	19	1792.59	19	1,576.81	23	2141.259	19	1491.2923	30.4%	17	1,486.12	0.3%
R103	15	1553.58	15	1,316.31	16	1651.6	15	1245.2884	24.6%	13	1,292.68	-3.7%
R104	12	1283.22	11	1,061.90	14	1419.203	12	1027.7393	27.6%	9	1,007.24	2.0%
R105	15	1534.40	15	1,455.08	22	1844.607	16	1401.7127	24.0%	14	1,377.11	1.8%
R106	15	1457.51	14	1,292.28	17	1767.745	14	1280.718	27.6%	12	1,251.98	2.3%
R107	13	1336.79	12	1,174.00	15	1584.442	12	1130.3644	28.7%	10	1,104.66	2.3%
R108	10	1174.06	9	1,030.87	12	1313.324	12	1006.2199	23.4%	9	960.88	4.7%
R109	14	1423.01	13	1,284.32	19	1714.363	14	1189.11	30.6%	11	1,194.73	-0.5%
R110	12	1332.66	13	1,205.48	17	1562.29	13	1136.9781	27.2%	10	1,118.59	1.6%
R111	13	1344.17	13	1,239.26	14	1604.695	13	1112.9772	30.6%	10	1,096.72	1.5%
R112	11	1167.79	11	1,059.78	12	1302.659	11	1001.0385	23.2%	9	982.14	1.9%

Table 10: R1 Solutions

5.6.4 R2

The environment in R2 which consist of a combination of random stops and low capacity constraint effect, resulted in better results through the use of more vehicles. The initial solution also display the trend to use more vehicles to get a shorter distance travelled.

	Tabu				New					
Prob	Initial Solution		Improvement		Initial Solution		Improvement			Best Published	
R201	5	1633.98	4	1,335.55	8	1,535.61	8	1,172.01	23.7%	4 1,252.37	-6.4%
R202	5	1570.04	4	1,200.26	8	1,363.80	8	1,102.86	19.1%	3 1,191.70	-7.5%
R203	4	1325.15	3	972.59	7	1,280.42	7	934.83	27.0%	3 939.54	-0.5%
R204	3	1054.39	3	842.54	6	1,075.00	5	771.54	28.2%	2 825.52	-6.5%
R205	4	1461.61	3	1,133.02	5	1,396.79	7	1,023.90	26.7%	3 994.42	3.0%
R206	3	1358.68	3	985.94	6	1,221.69	5	974.70	20.2%	3 906.14	7.6%
R207	3	1205.44	3	948.50	6	1,117.92	5	839.83	24.9%	2 893.33	-6.0%
R208	3	908.49	2	845.94	6	973.85	4	845.47	13.2%	2 726.75	16.3%
R209	4	1260.75	4	930.43	6	1,160.23	6	909.86	21.6%	3 909.16	0.1%
R210	4	1384.77	3	1,019.45	6	1,220.80	7	948.19	22.3%	3 939.34	0.9%
R211	3	1080.89	3	862.42	5	1,133.56	5	838.96	26.0%	2 892.71	-6.0%

Table 11: R2 Solutions

5.6.5 RC1

The RC problems might be the closest representation to real world environments. It represents an environment that is not predictable and that contains areas with patterns in the environment. The algorithm will benefit from the clustered areas and the pheromone trail that is influenced for used during improvement will assist in a balanced diversification strategy. Results with the cost function utilised more vehicles in certain problems which produced better results.

	Tabu				New					
Prob	Initial Solution		Improvement		Initial Solution		Improvement			Best Published	
RC101	16	1929.02	16	1,742.62	23	2,159.95	17	1,704.72	21.1%	14 1,696.94	0.5%
RC102	15	1789.29	15	1,625.30	23	1,985.22	15	1,535.30	22.7%	12 1,554.75	-1.3%
RC103	13	1613.99	13	1,403.99	18	1,790.02	12	1,327.32	25.8%	11 1,261.67	5.2%
RC104	12	1363.74	12	1,212.92	15	1,506.09	12	1,212.79	19.5%	10 1,135.48	6.8%
RC105	16	1805.33	16	1,706.53	21	2,040.03	17	1,576.19	22.7%	13 1,629.44	-3.3%
RC106	14	1581.39	14	1,502.00	17	1,873.54	14	1,483.42	20.8%	11 1,424.73	4.1%
RC107	13	1607.96	12	1,318.22	15	1,728.79	12	1,300.15	24.8%	11 1,230.48	5.7%
RC108	12	1340.10	12	1,240.27	13	1,438.51	11	1,164.81	19.0%	10 1,139.82	2.2%

Table 12: RC1 Solutions

5.6.6 RC2

The RC2 problems display the same trend as with the R2 problem, mainly because of the use of more vehicles. The multi-start initial approach allows best solutions to use more vehicles than the best registered initial solution. Most of the solution results display better than published because of the use of the vehicles.

			Tabu		New				
Prob	Initial Solution		Improvement		Initial Solution		Improvement		Best Published		
RC201	5	2131.14	4	1,474.86	9	1,856.60	8	1,317.55	29.0%	4	1,406.91	-6.4%
RC202	5	1943.42	4	1,298.28	10	1,700.06	8	1,148.13	32.5%	3	1,367.09	-16%
RC203	4	1595.85	3	1,081.34	8	1,534.76	7	985.91	35.8%	3	1,049.62	-6.1%
RC204	3	1184.48	3	883.53	5	1,107.90	5	829.57	25.1%	3	798.41	3.9%
RC205	6	1940.44	5	1,311.93	8	1,766.76	9	1,180.35	33.2%	4	1,297.19	-9.0%
RC206	4	1595.74	4	1,162.03	8	1,480.07	7	1,121.70	24.2%	3	1,146.32	-2.1%
RC207	4	1491.13	4	1,106.24	6	1,607.20	6	1,054.30	34.4%	3	1,061.14	-0.6%
RC208	3	1275.21	3	920.17	5	1,106.80	4	858.56	22.4%	3	828.14	3.7%

Table 13: RC2 Solutions

5.7 Summary

The result indicates that the algorithm is giving consistent results across all the different problems. The implementation of the Solomon cost function depends solely on the distance, which resulted in answers that has beaten the best published.

The number of vehicles used is more in these instances. The function has been left in this state to indicate the sensitivity of the algorithm on the adaptive object model. i.e. the implementation of the cost and constraint classes by the implementer. This flexibility cannot be achieved in a domain orientated implementation.

The flexibility of the solution can now be exploited and by altering only the cost function to incorporate the cost of the use of another vehicle.

6 PARALLEL IMPLEMENTATION

6.1 Overview

This chapter provides a brief overview of the adaption of the ASAO algorithm for parallel use. The objective is mainly to gain speed improvement with the least complexity in the adaption. The initiation of the approach will assist in alternatives for future expansion.

The design of a parallel algorithm can be viewed as consisting of four stages - Partitioning, Communication Analysis, Granularity Control and Mapping. The following simple example illustrates some of the issues involved in each of the stages.

Consider the scenario: *n* answer-scripts have to be marked, each of which contains the answers to *m* questions. The scripts could be viewed as the data, and the marking process itself as the computation to be performed on it.

In order to design a parallel solution to this problem, it must first be decomposed into smaller tasks which can be executed simultaneously. This is referred to as the partitioning stage.

This can be done in one of two ways. Each script could be marked by a different marker - this would require *n* markers. Alternatively, marking each question could be viewed as a task. This would result in *m* such tasks, each of which could be tackled by a separate marker, implying that every script passes through every marker.

In the first approach, the data (scripts) is first decomposed and then the computation (marking) is associated with it. This technique is called domain decomposition.

In the second approach, the computation to be performed (marking) is first decomposed and then the data (scripts) is associated with it. This technique is called functional decomposition. The partitioning technique that will be chosen often depends on the nature of the problem.

Suppose one needs to compute the average mark of the n scripts. If domain decomposition was chosen, then the marks from each of the markers would be required. If the markers are at different physical locations, then some form of communication is needed, in order to obtain the sum of the marks.

The nature of the information flow is specified in the communication analysis stage of the design. In this case, each marker can proceed independently and communicate the marks at the end. However, other situations would require communication between two concurrent tasks before computation can proceed.

It may be the case that the time to communicate the marks between two markers is much greater than the time to mark a question. In which case, it is more efficient to reduce the number of markers and have a marker work on a number of scripts, thereby decreasing the amount of communication.

Effectively, several small tasks are combined to produce larger ones, which results in a more efficient solution. This is called granularity control. For example, k markers could mark n/k scripts each. The problem here is to determine the best value of k.

The mapping stage specifies where each task is to execute. In this example, all tasks are of equal size and the communication is uniform, so any task can be mapped to any marker. However, in more complex situations, mapping strategies may not be obvious, requiring the use of more sophisticated techniques.

6.2 Single thread environment

The implementation of the ASAO algorithm was done from basic principles in the C# language. A standard off the shelf desktop was used to do the development and testing. The most basic desktops contain multiple processors. This machine runs two independent processor cores in one physical package at the same frequency.

With the introduction of the multi-processors, and Hyper-Threading Technology for the desktop, threading is no longer within the exclusive domain of Server application developers. In a single-core or traditional processor the CPU is fed strings of instructions it must order, execute, then selectively store in its cache for quick retrieval. When data outside the cache is required, it is retrieved through the system bus from random access memory (RAM) or from storage devices. Accessing these slows down performance to the maximum speed the bus, RAM or storage device will allow, which is far slower than the speed of the CPU. The situation is compounded when multi-tasking. In this case the processor must switch back and forth between two or more sets of data streams and programs. CPU resources are depleted and performance suffers.

In a dual core processor each core handles incoming data strings simultaneously to improve efficiency. Just as two heads are better than one, so are two hands. Now when one is executing the other can be accessing the system bus or executing its own code.

To utilize a dual core processor, the operating system must be able to recognize multi-threading and the software must have simultaneous multi-threading technology (SMT) written into its code. SMT enables parallel multi-threading wherein the cores are served multi-threaded instructions in parallel. Without SMT the software will only recognize one core. The CPU usage in Figure 47 displays the normal implementation of the ASAO algorithm in a Windows environment.

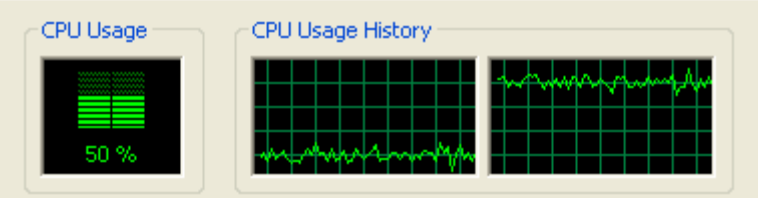

Figure 47: Desktop CPU usage single thread

The usage is clearly showing that the operating system does not balance processing across the two processors. The responsibility to implement a parallel solution is that of the developer. The following paragraphs step through the thinking of implementing the parallel ASAO.

6.3 Partitioning

Decomposing the algorithm into smaller tasks require an evaluation of the steps followed. We investigate the steps from a high level.

The first high-level step is reading the problem environment. This step requires basic information management and although there exist methods of improving reading data, we will not consider it as part of this design.

The second step consists of pre-calculations such as the calculation of the probability matrix and detection of neighbours. This still not part of the core algorithm and is done as a once off, but can easily be done in parallel as it is two non dependent separate processes. This is not currently of interest for us.

The third step is defined as the environment evaluation. This procedure is just a statistical review of the environment and does not contribute to the running time of the algorithm. It must be noted that when moving to a dynamic environment, this step might be an important tool in the continuous evaluation of the environment and can contribute to the effectiveness of the algorithm. It is not part of this study.

The core of the algorithm consists of two main steps, the initial solution and the improvement.

The initial solution approach is designed to provide multiple starting options through the use of multiple methods. This assists to diversify into the problemspace which assist in adapting to the environment sensitivity according to a specific method. In other implementations, an initial solution method is chosen depending on the environment and the improvement kicks-off from a single solution start. This lead to the initial solution being designed for the specific problem environment. The initial solution has been disregarded in the overall computational time. The impact of the initial solution approach in this study is significant enough to consider parallel processing. We define the parts as each different method that is used.

The improvement algorithm is the most complex to consider because of the impact it has on the computational time. The dependence on the number of ants to solve the process influences the efficiency of the parallel implementation. The ASAO algorithm provides a clear location for the partitioning, i.e. the individual ants that is responsible for the implementation of an improvement technique.

The challenge is to consolidate the results on the correct times and have the impact varies depending on the solution overlap and solution status.

6.4 Communication Analysis

It is clear from the previous paragraph that we will only consider the initial and improvement part of the solution for parallel implementation. This paragraph analyzes the communication required between the defined parts of the algorithm. The functional decomposition technique is used because we cannot clearly define the tasks inside the computation. The heuristic approach is dependent on randomness.

The initial solution approach spawns multiple scouts into the problem space to search for possible solutions. Scouts act individually and do not communicate with each other during the search process. They communicate their findings back to the queen who has to make a decision on the next action. The communication from the scouts to the queen contains all the solutions per scout and the method of the scout.

The queen can now evaluate the difference in quality for each scout and if necessary combine subparts from solutions to create new solutions. The communication required for the scouting does not depend on concurrent processes to share information.

The improvement phase's efficiency is more susceptible to communication between the processes. The heuristic type algorithm depends on memory structures to assist in decision making. In the single thread environment, the sequence can be determined and information can be passed over to the next ant on termination. Although complex, the implementation of parallel processing can contribute in speed of execution. With intelligent design, the speed can even be more approved by reducing the number of iterations because of communication between concurrent processes.

In the single thread environment, a process will finish before information is passed to the next action. If two processes based on different actions were executing concurrently, frequent messaging can abort the process that is either not efficient or running in the same area of the problem space.

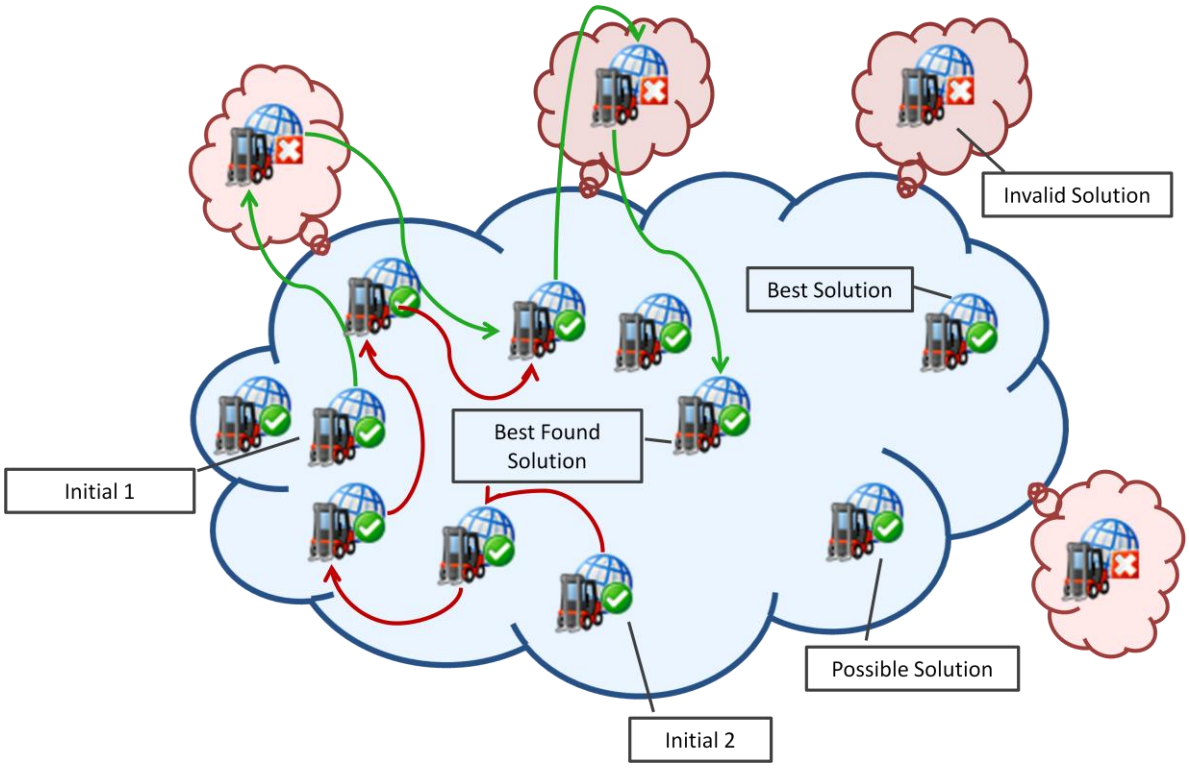

Figure 48: Multiple search paths

The scenario in Figure 48 depicts a high-level search pattern of the algorithm. The green line represents process 1 and the red line process 2. It is clear that the two processes are traversing in the same area of the problem space. The effort could be better spend on the area on the right hand side.

This is the main reason why traditional heuristic methods include a step for diversification. And the solution would still be found in traditional heuristic methods, but the time waist could have been detected earlier.

The proposed approach sounds simple enough, but the effort of the communication and comparing solutions at run-time is the major factors that will determine the success of this step. The environment knowledge must now play its part.

If the environment's topology is flat, as described in chapter 5, we know that there exist multiple solutions in a neighbourhood of moves. This environment normally requires a wide search and intensive local optimization. One process in a specific area is sufficient for the convergence because the local memory structures will guide the optimization the best. Comparing search patterns in this environment is difficult because solutions that exist in the

same area can have constructed totally different route structures. All because there exist a number of solutions in the neighbourhood that can be reached by one move.

If the environment topology consists of more mountains and valleys, the local optimization tends to converge to a locally optimal quite easily. The goal would be to terminate those processes that are climbing the same hill and only leave one to achieve the goal. Detecting these similarities should be easier in this environment, as the first few steps would quickly settle on a pattern of routes.

The proposed implementation of the communication channels consist of adding an additional layer of communication. The queen, who is in control, should place all ants in groups determined from the initial solutions. Efficiency can then be compared by

- Convergence speed – how does the algorithm climb the hill in relation to other processes? If the process is lacking in convergence compared to total cost, the area is most likely not suitable for a best solution.
- Route overlapping – stop sequence change and individual stop exchange between routes are the most effective moves close to the local optimum. Routes from the same area would overlap with a high percentage.
- Total cost – if the total cost of a process is not considered to be feasible, terminate.

To add to the efficiency of the communication, the importance of the efficiency parameter result should depend on the global status of the solution. The hybrid implementation based on simulated annealing provides information on the status of the solution.

In the initial phases of the improvement heuristic, convergence speed will be the dominant factor. The principle of initial solutions, that might be costly but still good because of its convergence in a specific area, applies. Total solution cost is always an important factor, while the route overlapping factor becomes dominant in the second part of the cooling down process.

6.5 Granularity Control

Determining the granularity of the implementation is not a straight forward task. The initial solution approach follows a strict implementation method, while the improvement step requires flexibility.

The initial solution is divided into 3 main methods: SIH on clustered chains, SIH, Constructive ant. These 3 methods can be executed in parallel without communication and speed problems.

The improvement step depends on the queen to control procedures. Part of the purpose of the queen is to decide the granularity, i.e. number of processes, through feedback from the processes running. As discussed in the previous paragraph, communication is done on 2 levels.

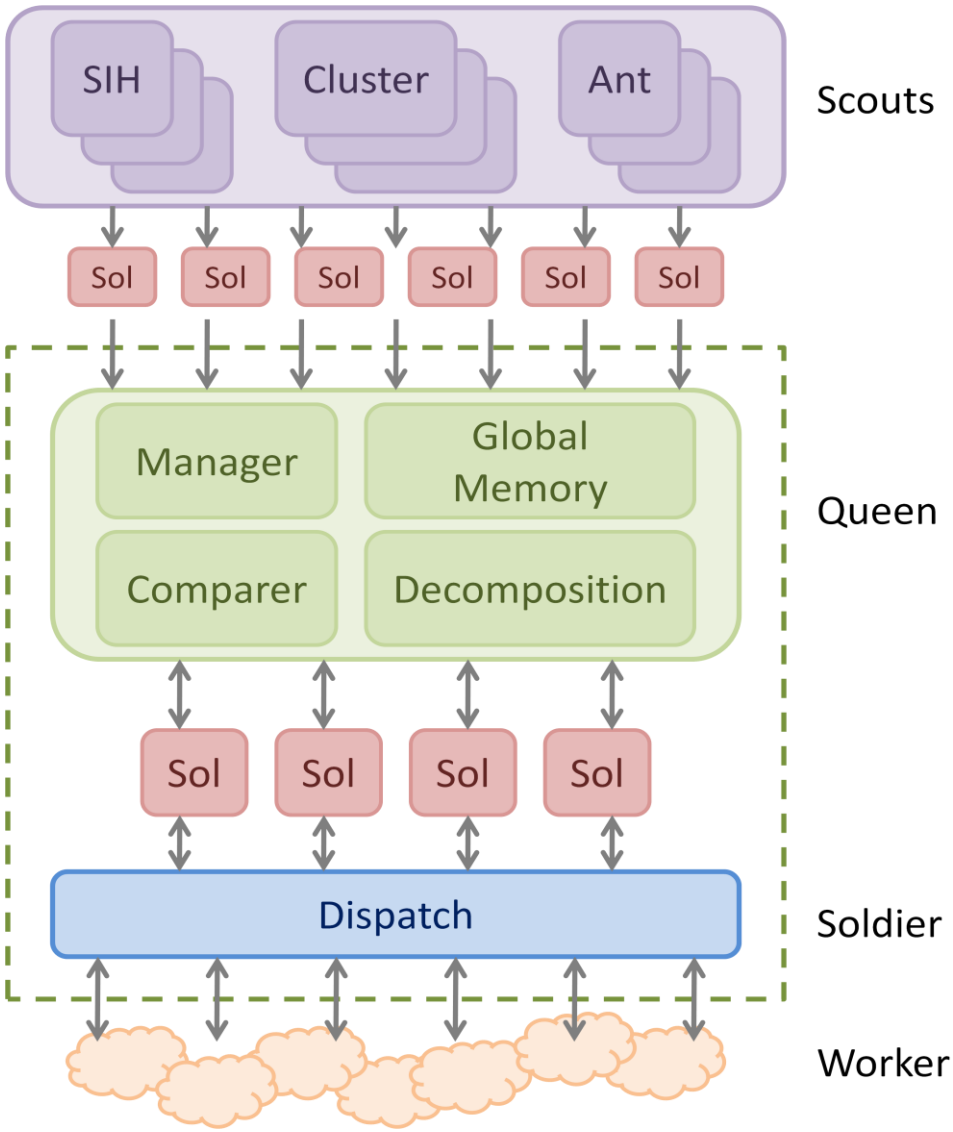

Figure 49: Organization of parallel search

The computational environment is not distributed computing where computers are connected with lower speed links. It is thus fine grain, closely connected architecture and the algorithms have been designed accordingly. The decomposition step, as illustrated in Figure 49, has the capability to decide on the number of solutions to use, thus determining the granularity of the system during run-time. Although the soldier decomposes the solution in more solutions, the parallel implementation will not be applied on that level.

6.6 Mapping

The mapping stage specifies where each task is to execute. In the initial solution stage, all tasks are viewed as equally important and are mapped to the same importance for process execution.

In the improvement stage, a sophisticated technique can be used. We define the following simple starting point of process priority:

1. Lowest cost – the solution with the lowest cost gets the primary consideration. The more complete initial solution method will result in a good solution for the majority of the time. The aim of this study to provide a feasible approach for business implementation, has a requirement that the algorithm to provide a solution as quick as possible. This priority will also set a realistic benchmark for other decisions.
2. Other – the rest of the solutions is next.
3. Scouts – it is important to keep scouting the area for uncharted territory. This step is only reached after a number of iterations, which indicates the complexity of the problem space. In this instance, the business will allow more iteration for a good solution.

The parallelization of the algorithm has many aspects to consider. The complexity of the unknown domain environment is an additional aspect to consider.

6.7 Parallel Ant System on Adaptive Objects

The previous paragraphs describe the steps to approach the parallelization of an algorithm. The partitioning paragraph recommended that the study focus on the initial solution and improvements stage for parallelization importance. This paragraph portrays the changes on the ASAO algorithm introduced in chapter 5.

The initial solution adaption is kept simple and can be interpreted in Figure 50. The algorithm spawns the initial solution methods in parallel. The expansion with new initial methods does not require a rework of the algorithm. Speed improvement on this fixed time part of the algorithm is notable and provides the option to reuse for diversification.

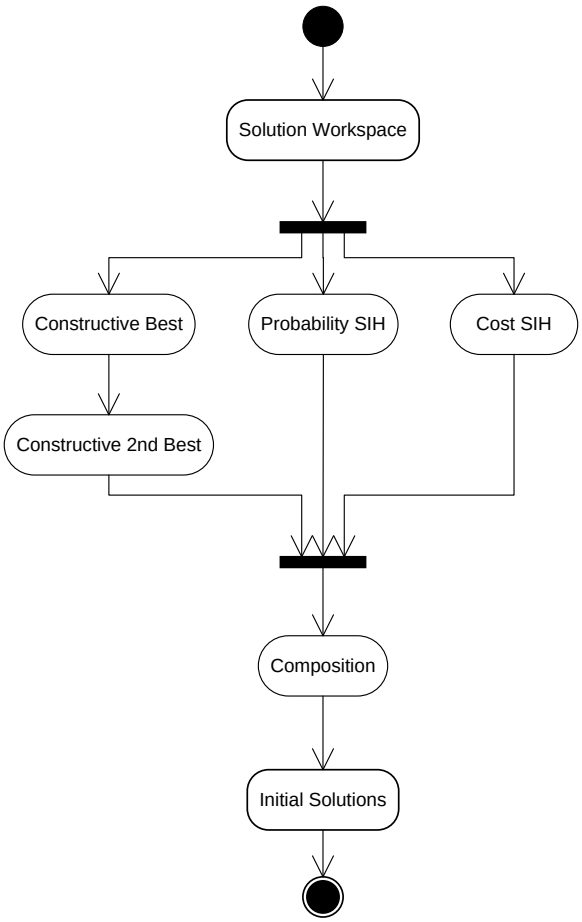

Figure 50: Initial PASAO

The result from the initial solution is evaluated and clearly marked to trace the origin for future references. A consolidated set of solutions is then passed on for further improvement. The two level control structures require partitioning of same type solutions to form the second level.

Figure 51 represents a high-level sequence diagram which indicates the purpose of each part of the system. It is important to note that once the algorithm has started, the queen spawns the process to the soldier. This entity continues to work until aborted by the queen. Figure 52 represents a more sequential flow of activities. From the diagrams, we can learn that the queen has a solution available at any time whilst the system is still executing. This approach provides flexibility which results in no adaption for each individual requirement. Best effort relates to the time allowed to execute.

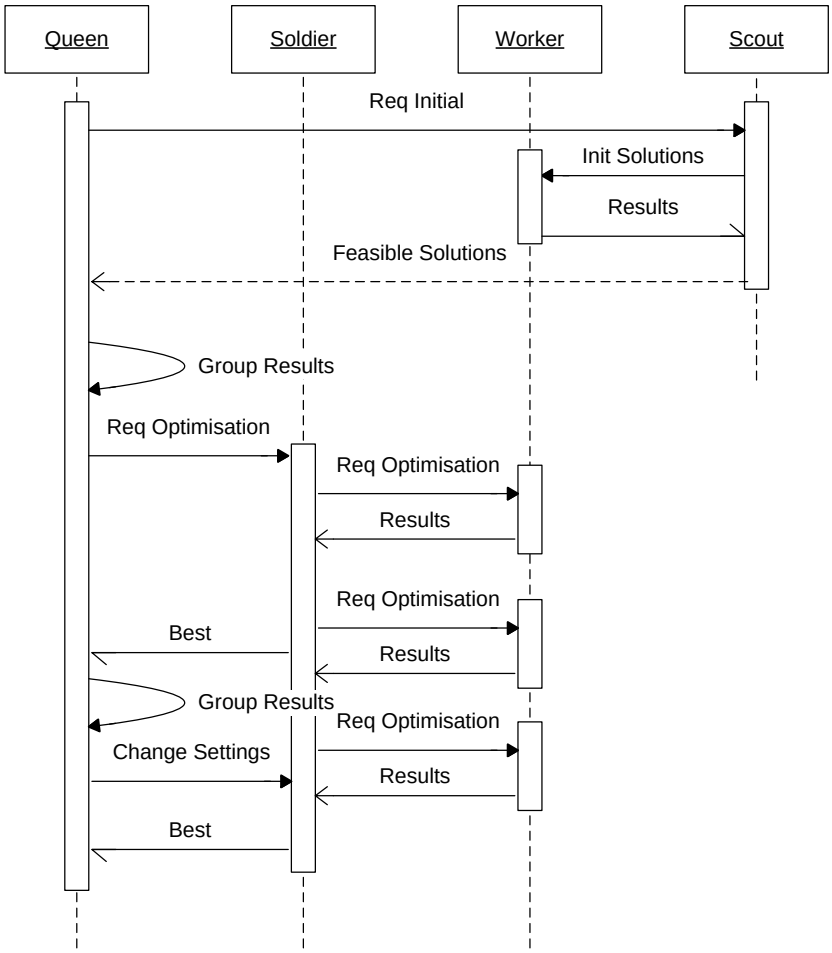

Figure 51: Sequence diagram for PASAO

The granularity of the solution is not known during design time for the improvement stage. The related diagrams must be interpreted accordingly. The same memory structure is used by the manager and the system. Each instance maintains its own structure's content.

Figure 51 display the call-back from the soldier to the queen as asynchronous. This is not clear in Figure 52, but can be interpreted that the 'join' action between the systems are not a dependency between the systems, but rather depends on the manager (queen). A 'join' command in C# force the main thread to wait until the thread has executed.

The straightforward transpose from the single threaded environment can be done in this way. This way, only utilization of multi-processor environment is achieved. The goal is to

simultaneously monitor the efficiency of the systems (soldiers) to be able to terminate where possible. Figure 52 must be interpreted with the asynchronous message return as described in Figure 51.

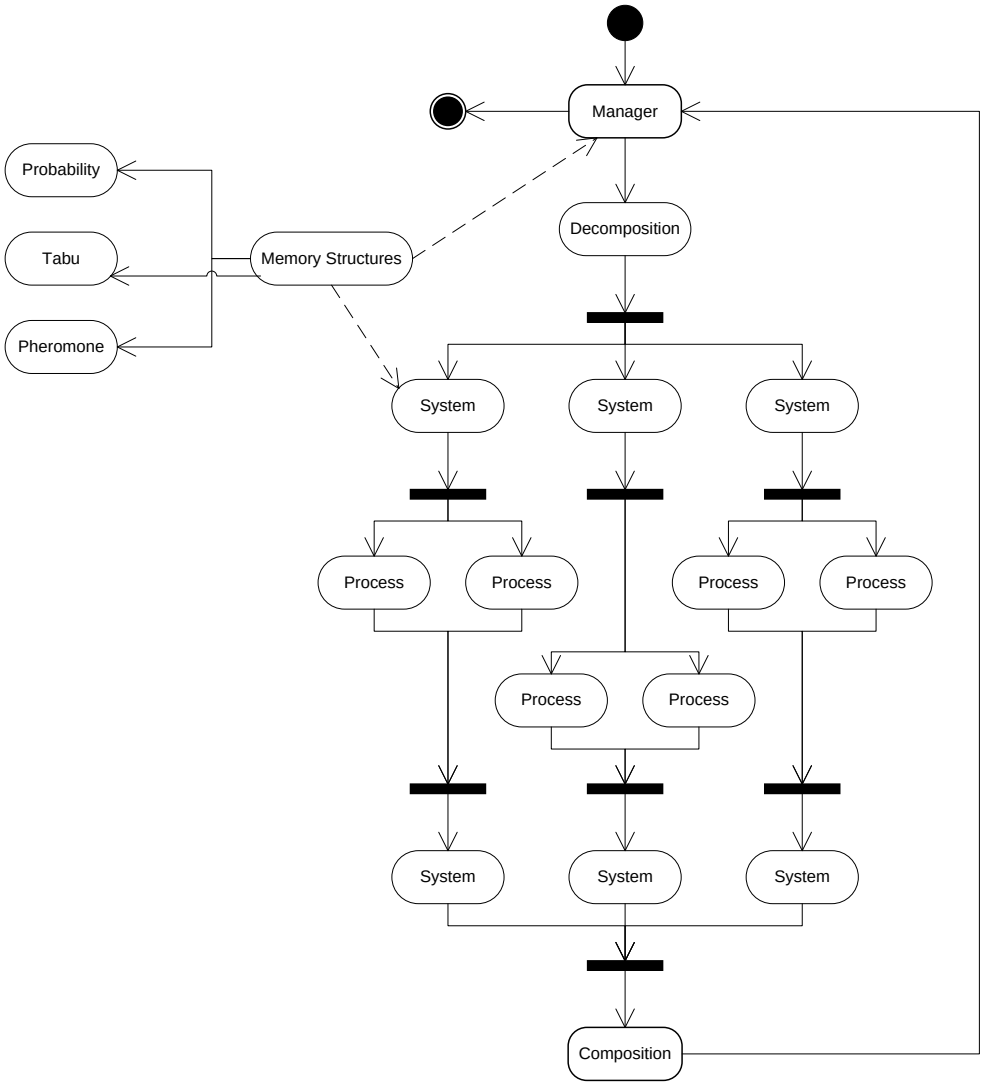

Figure 52: PASAO

6.8 Results

The primary goal for the parallel implementation is to utilize the multi-processor architect that is even available in today's personal computers. The solution results are expected to be the same or better as in a single threaded environment.

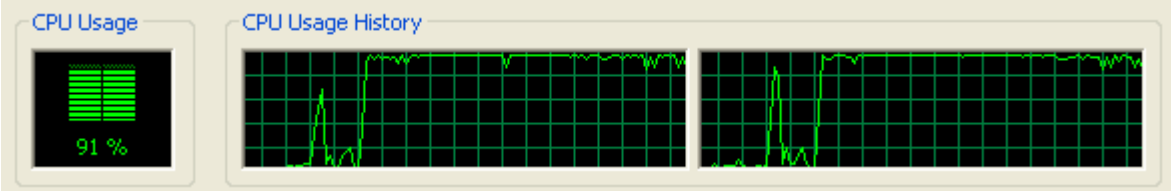

Figure 53: Desktop CPU usage multi-thread

The CPU processing depicted in Figure 53 clearly shows the improvement from the single threaded usage illustrated in Figure 47. It is also apparent where the initial solutions were generated, before the improvement started. The initial solution use a fixed time.

6.9 Summary

The parallelization of the ASAO algorithm has many possible benefits. This chapter explores some approaches. It designs some specific, not too complex, options for implementation. The advantage of utilizing multi-processes has immediate impact and cannot be seen as optional anymore, because of the availability of dual processor architecture in the basic computers on the market today.

Communication complexity between processes governs the design of the rest of the parallelization approach, i.e. partitioning, granularity and mapping. Combined with the adaptive object ambition of this study, this design should steer away from dependence on the domain knowledge. It was achieved by considering only what we know about the algorithm. As a result, each system contains the same memory structure and the queen has the decision on how to manipulate it. Inter process communication intervals are based on the Simulated Annealing methodology. The state of the cooling determines the communication required and the resulting actions to be taken.

Parallelization of VRP solutions is inevitable in today's implementation circumstances. The effect of implementation has various positive effects and can be clearly seen in the results.

7 CONLUSION

7.1 Overview

Logistics management is that part of the supply chain which plans, implements and controls the efficient, effective forward and reverse flow and storage of goods, services and related information. There are many opportunities to reduce total cost in supply chains. The prerequisites to many categories of cost savings, including supply chain management, are simplification, which corresponds to the basic lesson of Industrial Engineering 101: "Simplify before automating or computerizing"(Anderson, 2009).

The annual State of Logistics Report in the USA issued by the Council of Supply Chain Management Professionals shows that businesses spent a record $1.4 trillion on logistics in 2007. That's equal to 10.1 percent of Gross Domestic Product. It's the first time since 2000 that figure has exceeded 10 percent. Not surprisingly, most of the increases were related to fuel(Shulz, 2008). Transportation costs now account for 6.2 percent of nominal GDP. But capacity is permanently leaving the trucking industry as firms exit the market place and sell their equipment, often in foreign markets. The closing of Jevic Transportation, the nation's 71st-largest trucking company, is evidence that some truckers simply do not have adequate business plans in this era.

Solving different kinds of the Vehicle Routing Problem is an important area of Operations Research. Achieving improvement of only a small percentage may result in large. The cost of implementing a solution requires analysts and developers who understand the specific problem domain for the company. This initial cost of implementation is still a major drawback for companies to take the step towards implementing a solution.

Current study on the VRP focuses mostly on the efficiency and effectiveness indicators because it does not consider the problem as part of an enterprise system. This study reviews the importance of the indicators. Integration, both within and among cooperating enterprises, now comes first and is most important, providing the highest value. Dynamic integration creates the ability for many enterprises to participate within IT solution. Agility, the ability to react quickly, comes second.

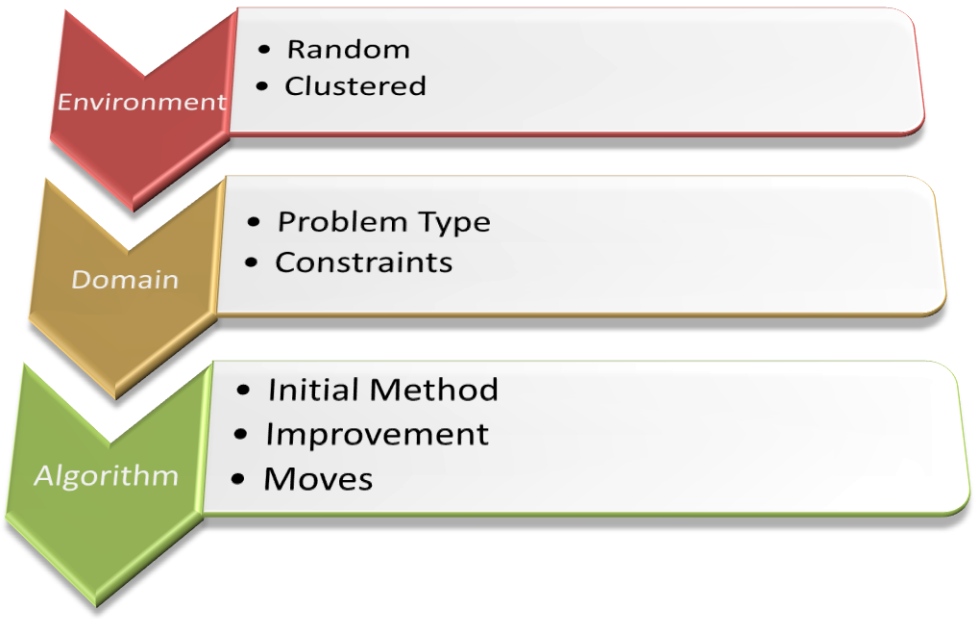

Figure 54: Traditional problem approach

This thesis addresses the problem of solving the VRP with variants through applying an adaptive solution which has no predefined knowledge of the problem environment. Current studies focus on solving VRP through identifying the type of problem beforehand and implement as solution for the specific type. Success is measured if an answer is *good enough* in a *reasonable time* for a problem where the user define the *constraint* and *cost* model.

7.2 Problem approach

The complexity of the VRP and all its variants has intrigued many OR researches. There exist numerous variants with numerous methods to solve them. The NP-hardness of the problem makes it then ideal playground for advance research. This study investigates the VRP problem with extraordinary requirements.

The innate ideas of solving the VRP consist of grouping stops according their geographical location. The expert will know that although the geographical location has a major impact on most real-life problems, the location is nothing else than a source for the cost calculation. The study seeks to steer away from the direct relation to the physical environment and view the

possibilities presented by the mathematical model. The inclusion of adaptive objects reduces the size of the mathematical model. The best-case approach in the initial environment to setup a cost matrix, contributes to the reducing of computational possibilities through the immediate elimination of impossible links.

The use of geographical independent clustering methods permits us to utilize traditional 'cluster first route second' methodologies. In combination with the pheromone trial, the results of clustering do not dominate the convergence. The memory structure for move types ensures that adaptiveness is also transferred to the algorithm. It is similar to solving the VRP with more than one algorithm.

7.3 Results

Results clearly indicate the accomplishment of the study. It must be acknowledged that the writer had a disproportionate advantage. The knowledge of designing the adaptive cost and constraint objects is strongly related to the problem domain as well as the internal algorithm use. This point toward the speed of the algorithm and not the efficiency.

The new definition on what qualifies as a good initial solution as well as the multi-start approach provides good coverage on the start of the algorithm. The addition of environment analysis influences the 'cooling' tempo of the convergence.

7.4 Summary

The design of a selective parallel heuristic algorithm for the Vehicle Routing Problem on an adaptive object model has been accomplish through integration of multiple methods. We conclude by stating that such a target would be impossible to achieve without hybrid methods. Adding the flexibility of determining run-time where and when to emphasize specific methods, contribute to the success of the design.

The study instigate numerous areas for further in-depth research of which the adaptive object model interface with the algorithm and the parallelization of the algorithm stands out the most. Both areas contain dependency on the memory structure used by the algorithm, as well as feedback from actions within the algorithm.

This study introduced an environment analysis model that has been dealt with from the static VRP problem viewpoint. The algorithm detects the type of environment during run-time. Further research on this method can assist in guiding the algorithm during the initial stage, which has enormous impact on the optimisation phase. A further development can include the re-evaluation of the environment when the optimisation memory structures have been set after a certain number of iterations.

A more complex problem such as the IRP (Inventory Routing Problem) can be implemented into the framework. The inventory routing problem involves the integration and coordination of two components of the logistics value chain: inventory management and vehicle routing. Inventory is required to provide cover against variability in demand, supply and the movement of products. Inventory is generally present in supply chain to ensure product availability. The inventory replenishment requirement is dependent on the time of visit, which is determined by the VRP.

A framework is a basic conceptual structure used to solve or address complex issues. A software framework is an abstraction in which common code providing generic functionality can be selectively overridden or specialized by user code providing specific functionality. This study introduces a framework for solving various types of complex VRP problems. A more in-depth study can be done on the differences between a framework base solution and a direct solution approach.

The parallel design discussed in the previous chapter is only the beginning of an important field of study for the VRP. There exist numerous approaches for parallel implementation from a computer scientist view. The multi-level usage of memory structures can be further expanded to fit the parallel approach. Problem granularity consists of the decision to breakdown the problem in smaller problems that can each be solved on its own and in parallel. Dynamic decision making on the feasibility of a granular level adds to the challenge and possibilities of parallel algorithms.

The methods utilised for this study can be applied on other problems and is not exclusive to the VRP only. The VRP provide a complex problem with multiple combinations that serves as a high-quality environment for research of new methods. This study steps towards an enterprise view for the VRP without sacrificing quality of the solution. This new approach opens up new possibilities.

8 BIBLIOGRAPHY

1. Albus, J.S. (2002) '4D/RCS A Reference Model Architecture for Intelligent Unmanned Ground Vehicles', Proceedings of the SPIE AeroSense Session on Unmanned Ground Vehicle Technology, Orlando.
2. Anderson, D.M. (2009) *Build to Order and Mass Customization*, Build to Order Consulting.
3. *Artificial Intelligence* (2008), [Online], Available: http://en.wikipedia.org/wiki/Artificial_intelligence.
4. Badeau, P., Gendreau, M., Guertin, F., Potvin, J.-Y. and Taillard, E. (1997) 'A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows'.
5. Bai, R., Burke, E.K., Gendreau, M. and Kendall, G. (2007) 'A Simulated Annealing Hyper-heuristic: Adaptive Heuristic Selection for Different Vehicle Routing Problems', Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA), Paris, France, 67-70.
6. Baker, B.M. and Ayechew, M.A. (2003) 'Agenetic algorithm for the vehicle routing problem', *Computers and Operations Research*, vol. 30, pp. 787-800.
7. Barbarosoglu, G. and Ozgur, D. (1999) *A tabu search algorithm for the vehicle routing problem. Computers and Operations Research, vol. 26, pp. 255-270.*.
8. Barbucha, D. and Jedrzejowicz, P. (2007) 'An agent-based approach to VRP', vol. 26.
9. Berger, J. and Barkaoui, M. (2002) *A Memetic Algorithm for the Vehicle Routing Problem with Time Windows*, Presented at the 7th International Command and Control Research and Technology Symposium.
10. Berger, J. and Barkaoui, M. (2003) 'A Route-directed Hybrid Genetic Approach for the Vehicle Routing Problem with Time Windows', *INFOR*, vol. 41, pp. 179--194.
11. Berger, J., Barkaoui, M. and Bräysy, O. (2002) 'A Parallel Hybrid Genetic Algorithm for the Vehicle Routing Problem with Time Windows'.

12. Berger, J., Salois, M. and Begin, R. (1998) *A Hybrid Genetic Algorithm for the Vehicle Routing Problem with Time Windows*, Vancouver: Lecture Notes in Artificial Intelligence 1418, AI'98, Advances in Artificial Intelligence.

13. Bianchessi, N. and Righini, G. (2007) 'Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery.', *Computers & Operations Research*, vol. 34, pp. 578-594.

14. Bianchi, L. and Mastrolilli, M. (2004) 'Research on the Vehicle Routing Problem with Stochastic Demand'.

15. Birattari, M. and Manfrin, M. (2004) 'Research on the Vehicle Routing Problem with Stochastic Demand'.

16. Blanton, J.L. and Wainwright, R.L. (1993) 'Multiple Vehicle Routing with Time and Capacity Constraints using Genetic Algorithms', Fifth International Conference on Genetic Algorithms.

17. Bonabeau, E., Dorigo, M. and Theraulaz, G. (1999) *Swarm Intelligence: from natural to artificial systems*, Oxford University Press US.

18. Bowerman, R., Calamiand, P. and Brent Hall, G. (1994) 'The spacefilling curve with optimal partitioning heuristic for the vehicle routing problem', *European Journal of Operational Research*, vol. 76, p. 128–142.

19. Bowers, M., Noon, C.E. and Thomas, B. (1996) 'A parallel implementation of the TSSP', vol. Computer Operations Research 23, no. 7.

20. Bozkaya, B., Erkut, E. and Laporte, G. (2003) 'A tabu search heuristic and adaptive memory procedure for political districting', *European Journal of Operational Research*, vol. 144, no. 1, pp. 12-26.

21. Bräysy, O., Berger, J. and Barkaoui, M. (2000) 'A New Hybrid Evolutionary Algorithm for the Vehicle Routing Problem with Time Windows', Presented in Route 2000 Workshop, Skodsborg, Denmark.

22. Bräysy, O. and Gendreau, M. (2005) 'Vehicle Routing Problem with Time Windows, Part II: Metaheuristics', *Transportation Science*, vol. 39, no. 1, pp. 119-139.

23. Buhl, J., Gautrais, J., Sole, R.V., Kuntz, P., Valverde, S., Deneubourg, J.-L. and Theraulaz, G. (2004) 'Efficiency and robustness in ant networks of galleries', vol. 42.

24. Bullnheimer, B., Hartl, R.F. and Strauss, C. (1997) 'Applying the Ant System to the Vehicle Routing Problem', 2nd International Conference on Metaheuristics, Vienna.

25. Campos, M., Bonabeau, E., Theraulaz, G. and Deneubourg, J.-L. (2001) 'Dynamic Scheduling and Division of Labor in Social Insects'.

26. Carlton, W.B. (1995) *A Tabu Search Approach to the General Vehicle Routing Problem.*, Ph.D. thesis, University of Texas, Austin, U.S.A.

27. Choi, E. and Tcha, D.W. (2005) 'A column generation approach to the heterogeneous fleet vehicle routing problem', vol. Computers & Operations Research 34.

28. Christofides, N., Mingozzi, A. and Toth, P. (1981) 'Exact algorithms for the vehicle routing problem based on spanning tree and shortest path relaxations. ', *Mathematical Programming*, vol. 20, pp. 255-282.

29. Clarke, G. and Wright, J.W. (1964) 'Scheduling of vehicles from a central depot to a number of delivery', *Operations Research*, vol. 12, pp. 568-581.

30. Colorni, A., Dorigo, M. and Maniezzo, V. (1991) 'Distributed Optimization by Ant Colonies', European Conference On Artificial Life, Paris, France, 134-142.

31. Cordeau, J.-F. and Laporte, G. (2002) 'Tabu Search Heuristics for the Vehicle Routing Problem'.

32. Cuesta-Cañada, A., Garrido, L. and Terashima-Marín, H. (2005) 'Building Hyper-heuristics Through Ant Colony Optimization for the 2D Bin Packing Problem', in *Knowledge-Based Intelligent Information and Engineering Systems*, Berlin / Heidelberg: Springer.

33. Cummings, N. (2000) *A BRIEF HISTORY OF TSP*, June, [Online], Available: http://www.orsoc.org.uk/about/topic/news/article_news_tspjune.htm.

34. Dantzig, G.B., Fulkerson, D.R. and Johnson, S.M. (1959) 'On a linear programming, combinatorial approach to the travelling salesman problem', *Operations Research*, vol. 7, pp. 59-66.

35. Dantzig, G.B. and Ramser, J.H. (1959) 'The truck dispatching problem', vol. 6, no. 1.

36. Dawkins, R. (1976) *The Selfish Gene*, Oxford, England: Oxford University Press.

37. De Magalhaes, J.M. (2006) 'Dynamic VRP in pharmaceutical distribution'.

38. Deneubourg, J.-L., Aron, S., Goss, S. and Pasteels, J.M. (1990) 'The Self-Organizing Exploratory Pattern of the Argentine Ant', vol. 3, no. 2.

39. Desrochers, M., Desrosiers, J. and Solomon, M.A. (1992) 'A new optimization algorithm for vehicle routing problem with time windows', *Operations Research*, vol. 40, p. 342–354.

40. Desrochers, M., Lenstra, J.K. and Savelsbergh, M.W.P. (1990) 'A classification scheme for vehicle routing and scheduling problems', *European Journal Opererations Research*, vol. 46, pp. 322-332.

41. Desrosiers, J., Sauve, M. and Soumis, F. (1988) 'Lagrangian relaxation methods for solving the minimum fleet size multiple traveling salesman problem with time windows.', *Management Science*, vol. 34, pp. 1005-1022.

42. Di Caro, G.A., Ducatelle, F. and Gambardella, L.M. (2008) 'Ant Colony Optimization for Routing in Mobile Ad Hoc Networks in Urban Environments', vol. Technical Report No. IDSIA-05-08.

43. Doerner, K., Hartl, R.F. and Reimann, M. (2001) 'A hybrid ACO algorithm for the Full Truckload Transportation'.

44. Dondo, R. and Cerdá, J. (2006) 'A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows', vol. Güemes 3450, 3000 Santa Fe, Argentina.

45. Dorigo, M. and Stutzle, T. (2004) *Ant colony optimization*, Bradford.

46. Dullaert, W. and Bräysy, O. (2003) 'Routing Relatively Few Customers per Route' Top.

47. Dussutour, A., Deneubourg, J.-L. and Fourcassie, V. (2005) 'Amplification of individual preferences in a social context the case of wall-following in ants', vol. 272.

48. Engelbrecht, A.P. (2007) *Computational Intelligence: An Introduction*, Wiley.

49. Ertöz, L., Steinbach, M. and Kumar, V. (2003) 'Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data'.

50. Ester, M., Kriegel, H.P., Sander, J. and Xu, X. (1996) 'A Density Based Algorithm for Discovering Clusters in Large An Object-Oriented Framework for Rapid

Development of Genetic Algorithms with Application to Operations Management and Vehicle Routingspatial Database with Noise'.

51. Estivill-Castro, V. (2002) 'Why so many clustering algorithms - A Position Paper', vol. 4, no. 1, p65.

52. Fisher, M. (1994) 'Optimal solution of vehicle routing problems using minimum K-trees', *Operations Research*, vol. 42, no. 4, pp. 626-642.

53. Fisher, M. and Jaikumar, R. (1981) 'A generalized assignment heuristic for vehicle routing', *Networks*, vol. 11, pp. 109-124.

54. Fisher, M., Jornsten, K. and Madsen, O. (1997) 'Vehicle routing with time windows: Two optimization algorithms', *Operations Research*, vol. 45, pp. 488-495.

55. Gambardella, L.M., Rizzoli, A.E., Oliverio, F., Casagrande, N., Donati, A.V., Montemanni, R. and Lucibello, E. (2003) 'Ant Colony Optimization for vehicle routing in advanced logistics systems'.

56. Gambardella, L.M., Taillard, E. and Agazzi, G. (1999) 'A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows', vol. Technical Report.

57. Garcia, B.-L., Potvin, J.-Y. and Rousseau, J.-M. (1994) 'A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Window Constraints.', *Computers and Operations Research*, vol. 21, pp. 1025-1033.

58. Gehring, H. and Homberger, J. (2000) 'A Parallel Hybrid Evolutionary Metaheuristic for the Vehicle Routing Problem with Time Windows'.

59. Gendrau, M., Laporte, G. and Potvin, J.Y. (1997) 'Vehicle routing: Modern heuristics.', *Local Search in Combinatorial Optimization.*, pp. 311-336.

60. Gendreau, M. and Bräysy, O. (2001) 'Tabu Search Heuristics for the Vehicle Routing Problem with Time Windows'.

61. Gendreau, M., Hertz, A. and Laporte, G. (1994) 'A tabu search heuristic for the vehicle routing problem', *Management Science*, vol. 40, pp. 1276-1290.

62. Gillet, B.E. and Miller, L.R. (1974) 'A heuristic algorithm for the vehicle dispatching problem', *Journal of Operations Research*, vol. 22, no. 4, pp. 340-349.

63. Glover, F.W. (1986) 'Future Paths for Integer Programming and Links to Artificial Intelligence', *Computers and Operations Research*, vol. 13, no. 5, p. 533–549.

64. Golden, B., Raghavan, S. and Wasil, E. (2008) *The Vehicle Routing Problem, Latest Advances and New Challenges*, Springer.

65. Golden, B., Wasil, E.A., Kelly, J.P. and Chao, I.M. (1998) 'The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results', in Crainic T, L.G. (ed.) *Fleet management and logistics*, Boston, MA: Kluwer.

66. Goss, S., Aron, S., Deneubourg, J.-L. and Pasteels, J.M. (1989) 'Self-organized shortcuts in the argentine ant', vol. 76, 579-581.

67. Halse, K. (1992) *Modeling and solving complex vehicle routing problems*, Ph.D. Dissertation no. 60, IMSOR, Technical University of Denmark.

68. *History of the TSP* (2007), January, [Online], Available: http://www.tsp.gatech.edu/history/index.html.

69. Homberger, J. and Gehring, H. (1999) 'Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows', *INFOR*, vol. 37, pp. 297-318.

70. Hwang, H.S. (2002) 'Design of supply-chain logistics system considering service level', vol. Computer and Industrial Engineering 43.

71. Jantzen, J. (1998) 'Neurofuzzy Modelling', vol. Tech. report no 98-H-874.

72. Jornsten, K., Madsen, O. and Sorensen, B. (1986) 'Exact solution of the vehicle routing and scheduling problem with time windows by variable splitting', *Research Report 5/86, Technical University of Denmark*.

73. Joubert, J.W. and Claasen, S.J. (2006) 'A sequential insertion heuristic for the initial solution to a constrained vehicle routing problem', vol. 22(1).

74. Kallehauge, B. (2008) 'Formulations and exact algorithms for the vehicle routing problem with time windows', *Computers and Operations Research*, vol. 35, no. 7, pp. 2307-2330.

75. Kaplan, S.J. (1984) 'The Industrialization of Artificial Intelligence: From By-line to Bottom Line', vol. 5, no. 2.

76. Kindervater, G.A.P. and Savelsbergh, M.W.P. (1997) 'Vehicle Routing: Handling Edge Exchanges' Chichester: Wiley.

77. Kolen, A., Rinnooy, A. and Trienekens, H. (1987) 'Vehicle routing with time windows. ', *Operations Research*, vol. 35, pp. 266-273.

78. Laporte, G. (2004) 'Vehicle-routing 15 years research'.

79. Laporte, G. (2007) 'What you should know about the vehicle routing problem', *Naval Research Logistics*, vol. 54, no. 8, pp. 811-819.

80. Laporte, G. and Semet, F. (2002) 'Classical heuristics for the capacitated VRP.', in Toth, P. and Vigo, D. *The Vehicle Routing Problem*, Philadephia, PA: SIAM Monographs on Discrete Mathematics and Applications.

81. Larsen, J. (1999) 'Parallelization of the Vehicle Routing Problem with Time Windows'.

82. Lieberherr, K. (1996) *Adaptive object-oriented software: The Demeter Method*, College of Computer Science, Northeastern University Boston.

83. Mailleux, A., Deneubourg, J.-L. and Detrain, C. (2000) 'How do ants assess food volume?', vol. 59.

84. Mailleux, A.-C., Deneubourg, J.-L. and Detrain, C. (2003) 'How does colony growth influence communication in ants?', vol. 50.

85. Marais, E. *Eugene Marais - Writer and Scientist*, [Online], Available: http://www.encounter.co.za/article/140.html.

86. Medaglia, A.L. and Gutierrez, E. (2005) 'An Object-Oriented Framework for Rapid Development of Genetic Algorithms with Application to Operations Management and Vehicle Routing'.

87. Mendoza, J.E., Castanier, B., Guéret, C., Medaglia, A.L. and Velasco, N. (2010) 'A memetic algorithm for the multi-compartment vehicle routing problem with stochastic demands', *Computers and Operations Research archive*, vol. 37, no. 11.

88. Millonas, M.M. (1992) 'Swarms, Phase Transitions, and Collective Intelligence', vol. Complex Systems Group, Theoretical Division and Center for Nonlinear Studies.

89. Moccia, L., Cordeau, J.-F. and Laporte, G. (2010) *An incremental tabu search heuristic for the generalized vehicle routing problem with time windows*, 27 January, [Online], Available: http://neumann.hec.ca/chairedistributique/common/gvrp.pdf [2 May 2010].

90. Montemanni, R., Gambardella, L.M., Rizzoli, A.E. and Donati, A.V. (2003) 'A new algorithm for a Dynamic Vehicle Routing Problem based on Ant Colony System'.

91. Moolman, A.J. (2004) *Design And Implementation Of An Integrated Algorithm For The Vehicle Routing Problem With Multiple Constraints*, Pretoria: University of Pretoria.

92. Moreira, A., Santos, M.Y. and Carneiro, S. (2005) 'Density based clustering algorithms'.

93. Morin, R.C. (2002) *dotNet Threading, Part I*, [Online], Available: www.kbcafe.com.

94. Moscato, P. and Cotta, C. (2005) 'Memetic Algorithms'.

95. Mutalik, P.P., Knight, L.R., Blanton, J.L. and Wainwright, R.L. (1992) 'Solving Combinatorial Optimization Problems Using Parallel Simulated Annealing And lParallel Genetic Algorithms'.

96. Nagy, G. and Salhi, S. (2004) 'Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries', vol. European Journal of Operational Research 162.

97. Nicolis, S.C. and Deneubourg, J.-L. (1999) 'Emerging Patterns and Food Recruitment in Ants an Analytical Study'.

98. Nilsson, N.J. (1998) *Artificial Intelligence: A new Synthesis*, Morgan Kaufmann.

99. *Office of National Statistics, News Release* (2006), [Online], Available: www.statistics.gov.uk.

100. Or, I. (1976) *Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking.* , Evanston, IL: Ph.D. Thesis, Department of Industrial Engineering and Management Sciences, Northwestern University.

101. Osman, I.H. (1993) 'Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem', *Annals of Operations Research*, vol. 41, pp. 421-451.

102. Park, H.-S., Lee, J.-S. and Jun, C.-H. (n.d) 'A K-means-like Algorithm for K-medoids Clustering and its Performance'.

103. Pasteels, J.M., Deneubourg, J.-L. and Goss, S. (1987) 'Self-Organization Mechanisms in Ant Societies (I) : trail recruitment to newly discovered food sources', vol. 54: 155-157.

104. Peterson, J.D. (2002) *Clustering Overview*.

105. Pisinger, D. and Ropke, S. (2005) 'A general heuristic for vehicle routing problems', vol. Computers & Operations Research 34.

106. Portha S, D.J.-L.D.C. (2004) 'How food type and brood influence foraging decisions of Lasius niger scouts', vol. 68.

107. Potvin, J.-Y. and Bengio, S. (1996) 'The Vehicle Routing Problem with Time Windows Part II: Genetic Search', *Informs Journal on Computing*, vol. 8, no. 2, pp. 165-172.

108. Potvin, J.-Y. and Naud, M.A. (2009) *Tabu Search with Ejection Chains for the Vehicle Routing Problem with Private Fleet and Common Carrier*, November, [Online], Available: https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2009-50.pdf [2 May 2010].

109. Potvin, J.-Y. and Rousseau, J.-M. (1993) 'A parallel route building algorithm for the vehicle routing and scheduling problem with time windows', *European Journal of Operational Research*, vol. 66, pp. 331-340.

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

111. Prins C (2001) 'A Simple and Efective Evolutionary Algorithm for the Vehicle Routing Problem'.

112. Prins, C. and Bouchenoua, S. (2002) *A Memetic Algorithm Solving the VRP, the CARP and General Routing Problems with Nodes, Edges and Arcs*, Presented at the Third Workshop on Memetic Algorithms, Granada, Spain.

113. Qili, Z. (1999) 'Heuristic Methods For Vehicle Routing Problem with Time Windows'.

114. Rego C (2001) 'Node-ejection chains for the vehicle routing problem Sequential and parallel algorithms', vol. Parallel Computing 27.

115. Reimann, M., Doerner, K. and Hartl, R.F. (2003) 'Analyzing a Unified Ant System for the VRP and Some of Its Variants'.

116. Rizzoli, A.E., Oliveiro, F., Montemanni, R. and Gambardella, L.M. (2004) 'Ant Colony Optimisation for vehicle routing problems: from theory to applications'.

117. Rochat, Y. and Taillard, R.E. (1995) 'Probabilistic diversification and intensification in local search for vehicle routing.', *Journal of Heuristics*, vol. 1, pp. 147-167.

118. Schulman, J. (August 2002) 'Governance and Management of Enterprise Architectures'.

119. Schulze, J. and Fahle, T. (1999) 'A parallel algorithm for the vehicle routing problem with time window constraints. ', *Annals of Operations Research*, vol. 86, pp. 585-607.

120. Shulz, J. (2008) *State of the Logistics Union*, [Online], Available: www.scdigest.com.

121. Solomon, M.M. (1987) 'Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints', *Operations Research*, vol. 35, pp. 254-265.

122. *Solomon-benchmark, 100-customers* (2010), 6 May, [Online], Available: http://www.sintef.no/Projectweb/TOP/Problems/VRPTW/Solomon-benchmark/100-customers/ [3 July 2010].

123. Sowa, J.F. (2000) *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks Cole Publishing Co., Pacific Grove, CA.

124. Taillard, E.D. (1999) 'A heuristic column generation method for the heterogeneous fleet VRP', vol. 33, no. 1.

125. Taillard, E.D., Gambardella, L.M., Gendreau, M. and Potvin, J. (2001) 'Adaptive memory programming: A unified view of metaheuristics', vol. 135, pp. 1-16.

126. Talbi, E.G., Hafidi, Z. and Geib, J.-M. (1998) 'Parallel adaptive tabu search for large optimization problems'.

127. Tan, K.C., Lee, L.H. and Ou, K. (2001) 'Hybrid Genetic Algorithms in Solving Vehicle Routing Problems with Time Window Constraints', *Asia-Pacific Journal of Operational Research 18*, vol. 18, pp. 121-130.

128. Thangiah, S.R., Osman, I., Vinayagamoorthy, R. and Sun, T. (1994) 'Algorithms for Vehicle Routing with Time Deadlines. ', *American Journal of Mathematical and Management Science's special issue: Vehicle Routing 2000: Advances in Time Windows, Optimality , Fast Bounds and Multi-Depot Routing*, vol. 13, no. 3-4, pp. 323-355.

129. Thompson, P.M. and Psaraftis, H.N. (1993) 'Cyclic Transfer Algorithm for Multivehicle Routing and Scheduling Problems', *Operations Research*, vol. 41, no. 5, pp. 935-946.

130. Toth, P. and Vigo, D. (1998) *The granular tabu search and its application to the VRP.*, Technical report, University of Bologna.

131. Toth, P. and Vigo, D. (2001) *The Vehicle Routing Problem*, Society for Industrial and Applied Mathematics Philadelphia, PA, USA.

132. Turban, E. and Aronson, J.E. (2001) *Decision Support Systems and Intelligent Systems*, Upper Saddle River: Prentice Hall.

133. Turing, A.M. (1950) 'Computing machinery and intelligence', *Mind*, vol. 59, pp. 433-460.

134. Valle, C.A., da Cunha, A.S., Mateus, G.R. and Martinez, L.C. (2009) 'Exact algorithms for a selective Vehicle Routing Problem where the longest route is minimized', *Electronic Notes in Discrete Mathematics*, vol. 35, pp. 133-138.

135. Van Schalkwyk, W.T. (2002) *An algorithm for the Vehicle Routing Problem with various side constraints*, University of Pretoria.

136. Winston, W.L. (1994) *Operations Research: Applications and Algorithms, Third Edition*, California.

137. Winston, P.H. and Prendergast, K.A. (1984) *The AI business: commercial uses of artificial intelligence*, Cambridge: Massachusetts Institute of Technology.

138. Xu, J. and Kelly, J.P. (1996) 'A Network flow-based Tabu Search heuristic for the Vehicle Routing Problem'.

139. Yeun, L.C., Ismail, W.R., Omar, K. and Zirour, M. (2008) 'Vehicle Routing Problem: Models and Solutions', *Journal of Quality Measurement and Analysis*, vol. 4, no. 1, pp. 205-218.

140. Yoder, J.W. and Razavi, R. (2000) 'Metadata and Adaptive Object-Models', in *Lecture Notes in Computer Science*, Berlin: Springer.

141. Yue, S., Li, P., Guo, J. and Zhou, S. (2004) 'Using Greedy algorithm: DBSCAN revisited II', vol. 5, no. 11.