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.

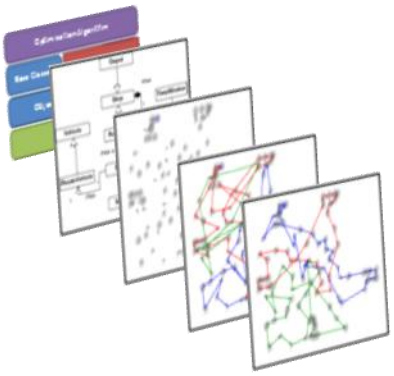

Figure 8: From modelling to solution

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.

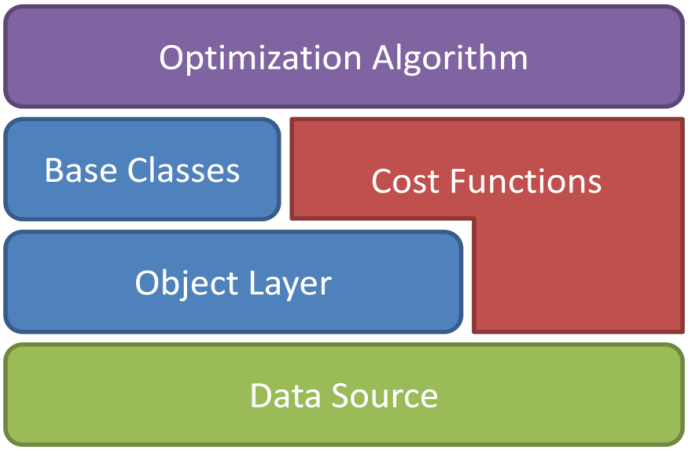

Figure 9: Solution component breakdown

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.

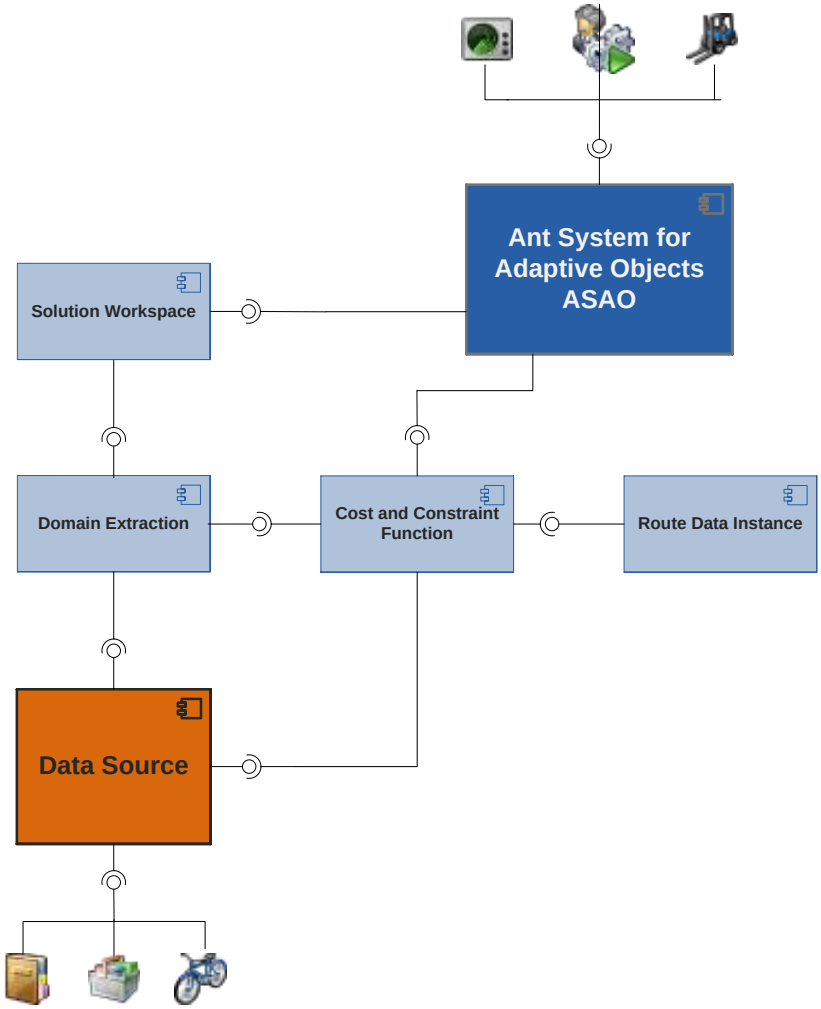

Figure 10: System logical overview

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.

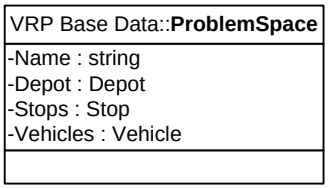

Figure 11: Problem Space Class

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.

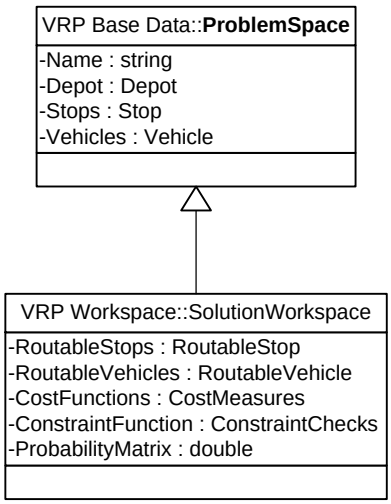

Figure 12: Solution Workspace

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.

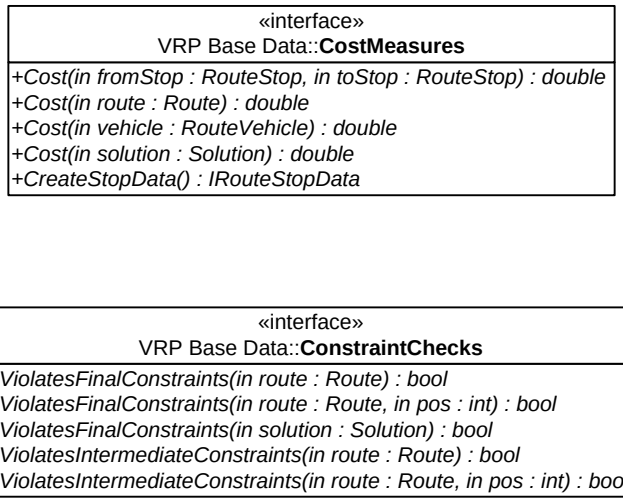

Figure 13: Cost and Constraint Interfaces

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:

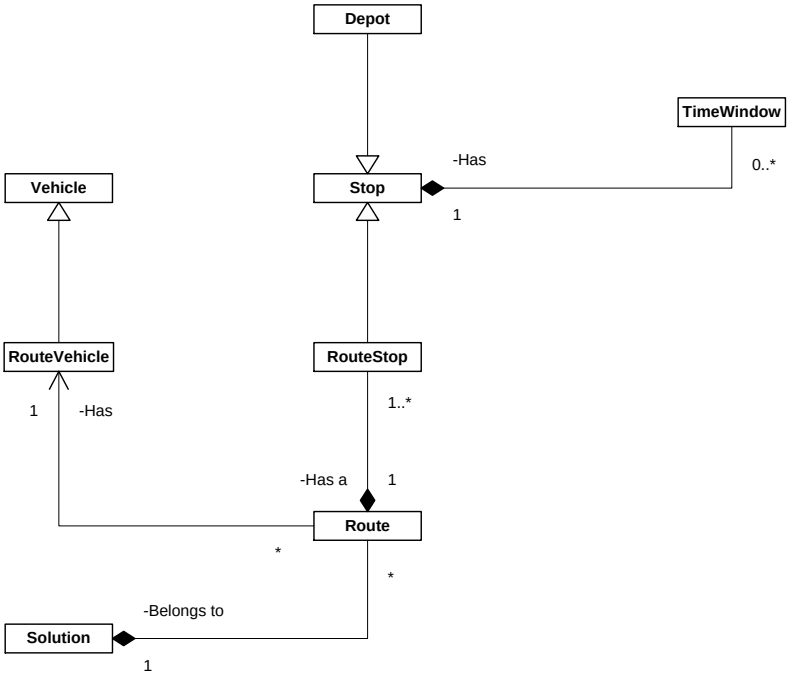

Figure 14: Basic domain class relations

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.

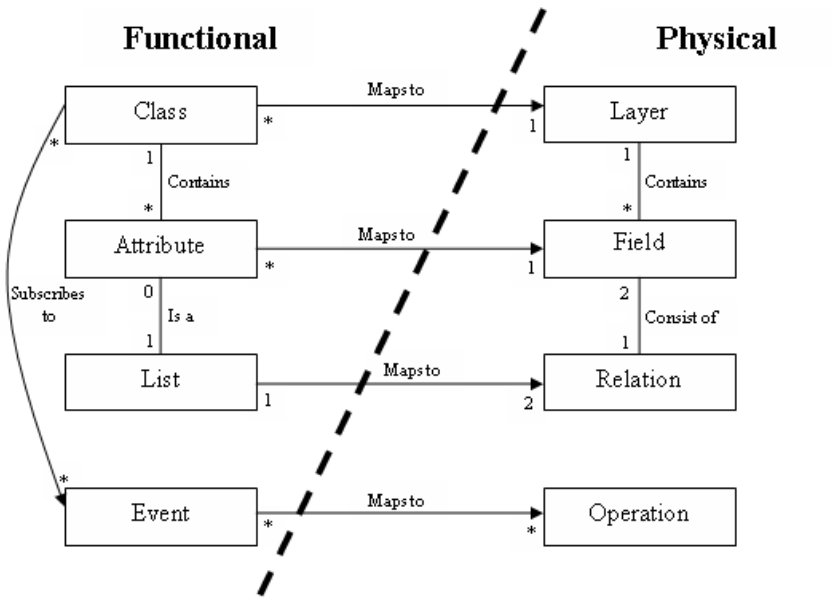

Figure 15: Meta mapping, functional to physical

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.

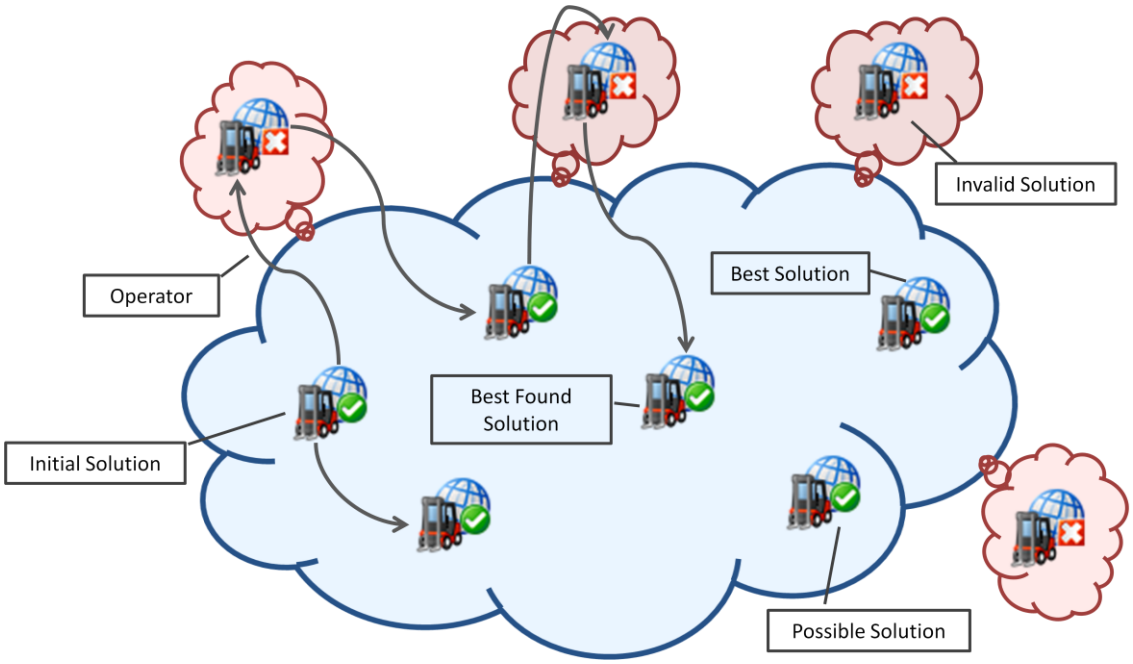

Figure 16: Solution Space

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).

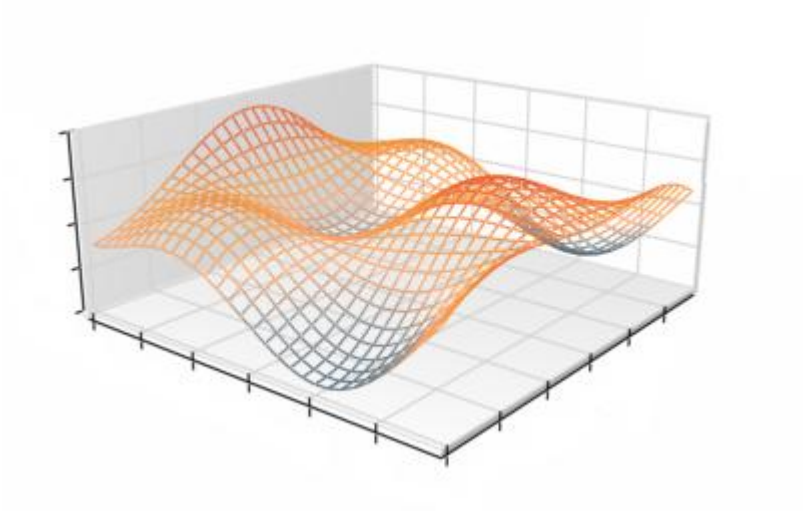

Figure 17: Solution Neighbourhood

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.

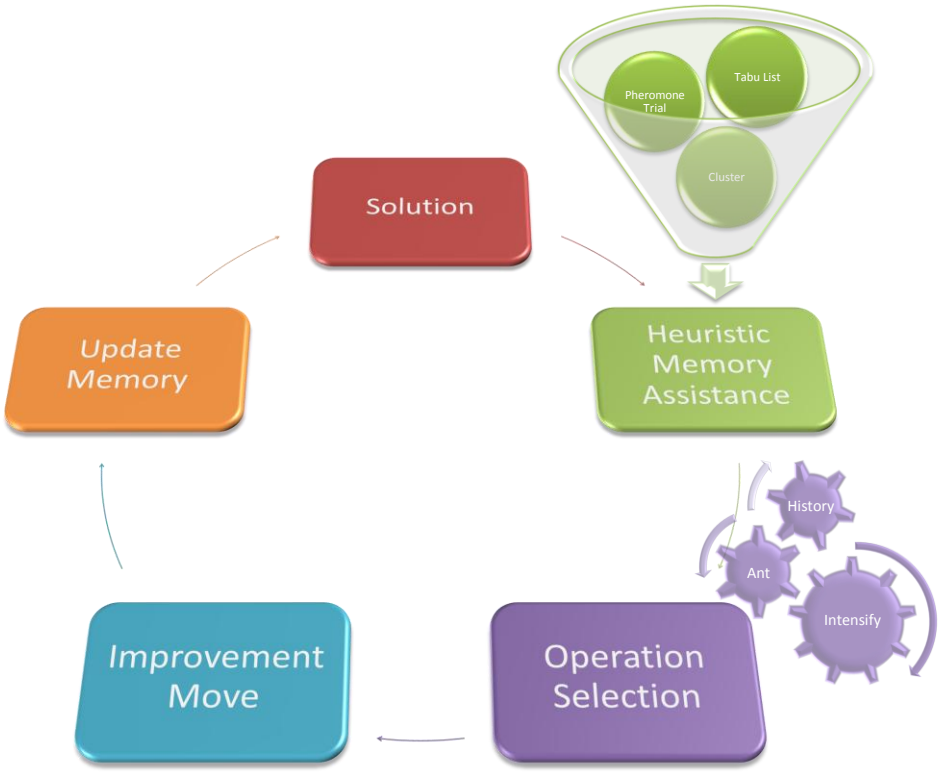

Figure 18: Solution approach overview

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.

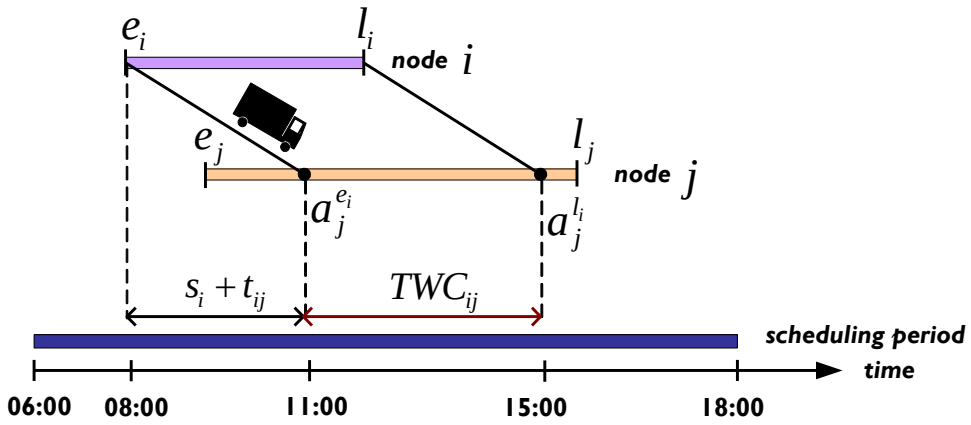

Figure 19: The basic TWC calculation - Scenario 0

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:

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

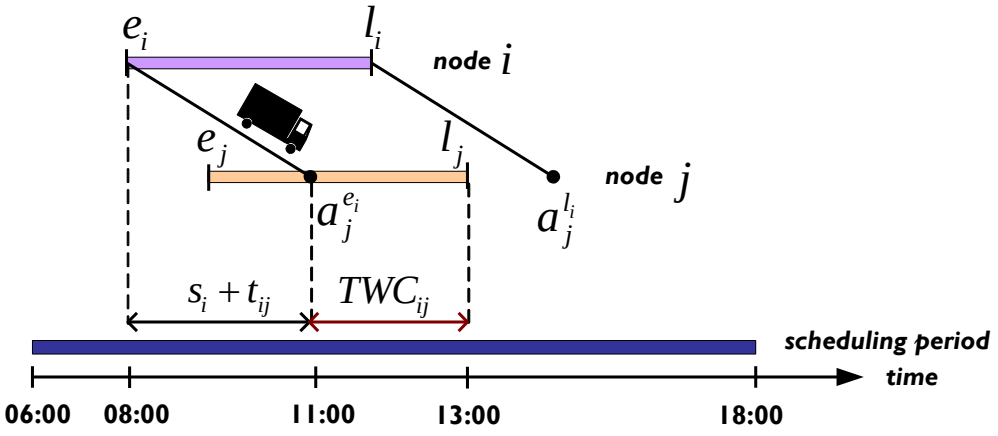

Figure 20: Scenario 1 TWC calculation

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:

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

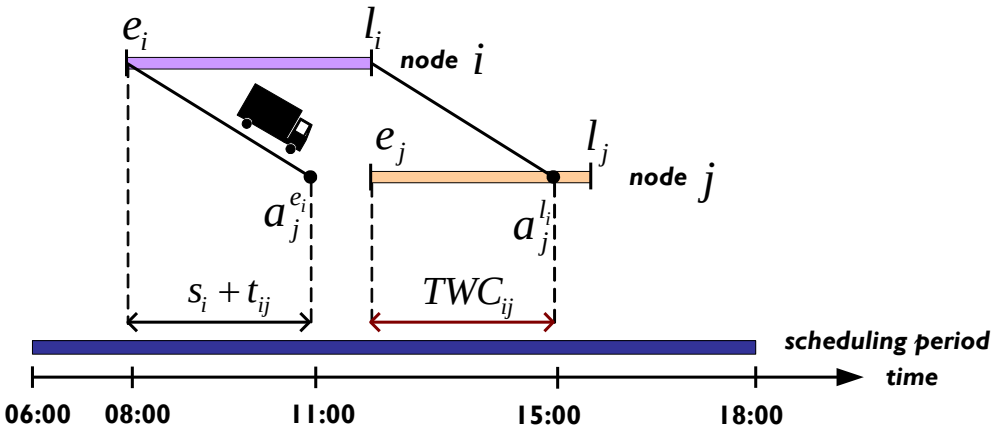

Figure 21: Scenario 2 TWC calculation

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:

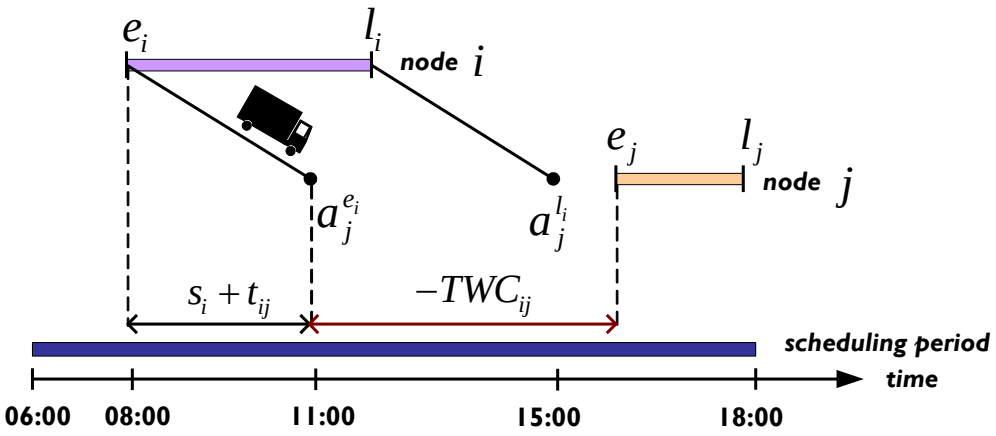

Figure 22: Scenario 3 TWC calculation

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.

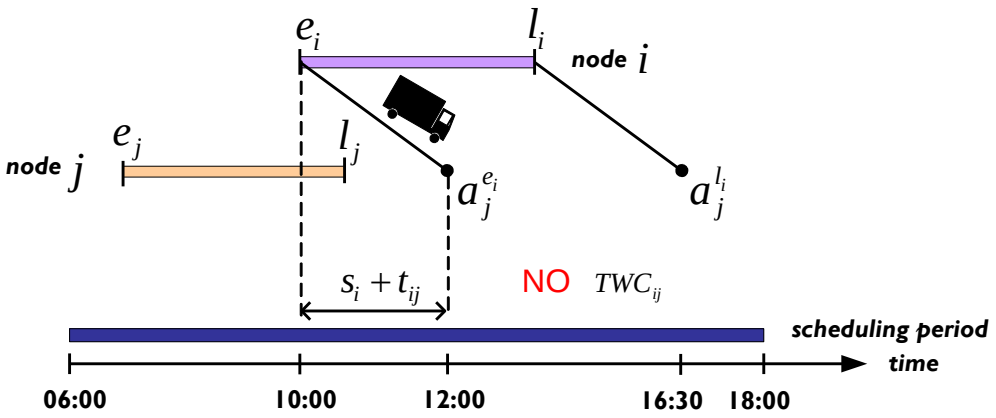

Figure 23: Scenario 4 - infeasible combination

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.

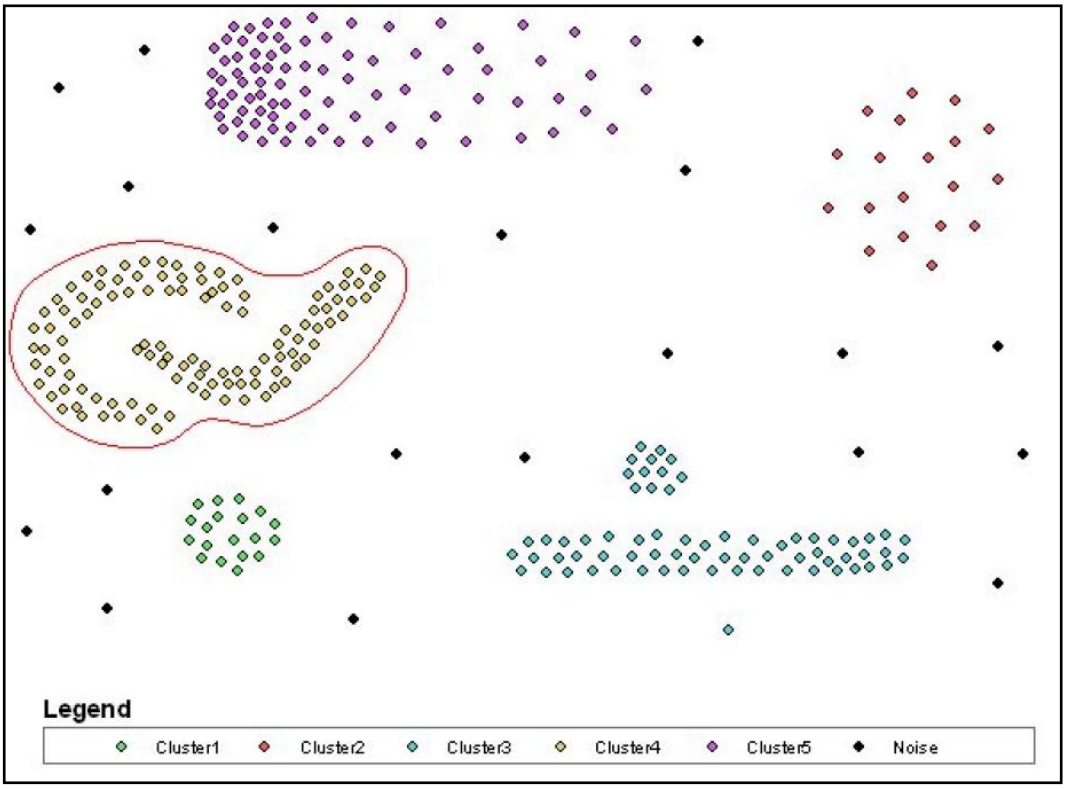

Figure 24: DBSCAN cluster

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.

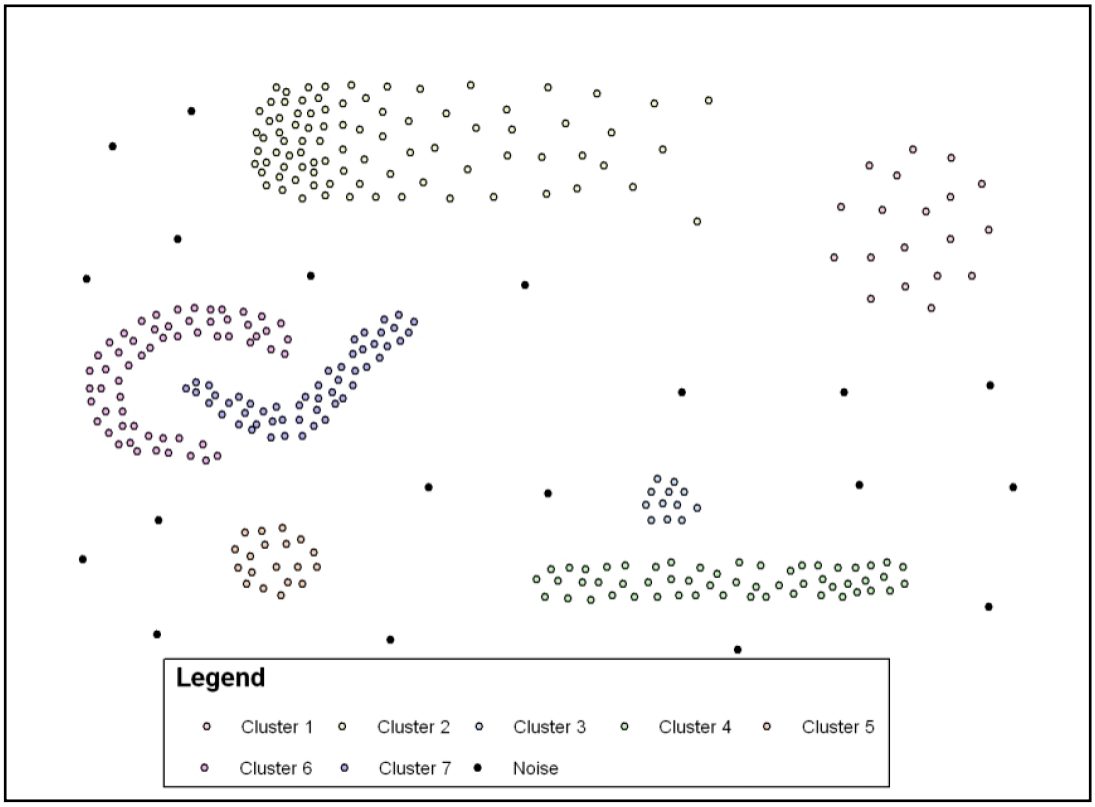

Figure 25: SNN cluster

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.

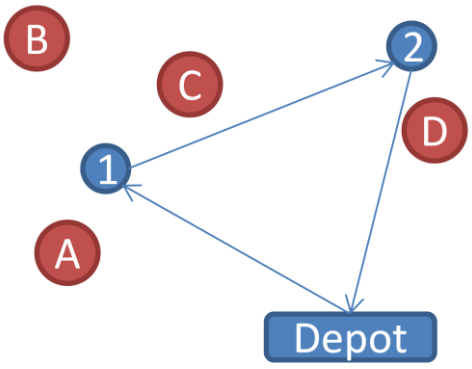

Figure 26: Adapted PFSIH

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.

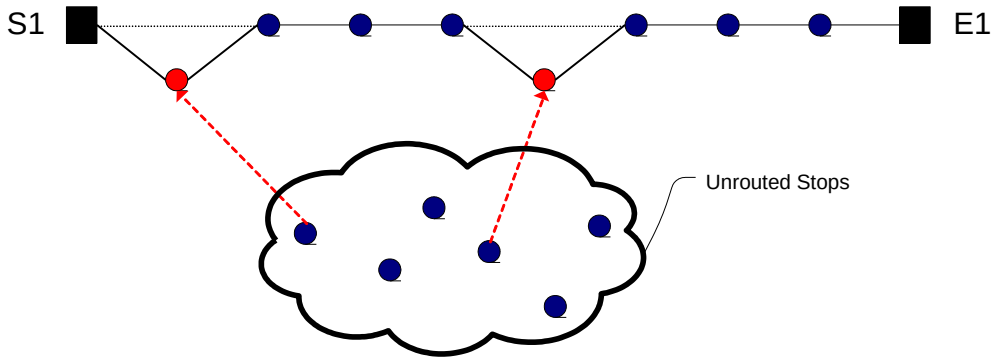

Figure 27: Insert Operation

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.

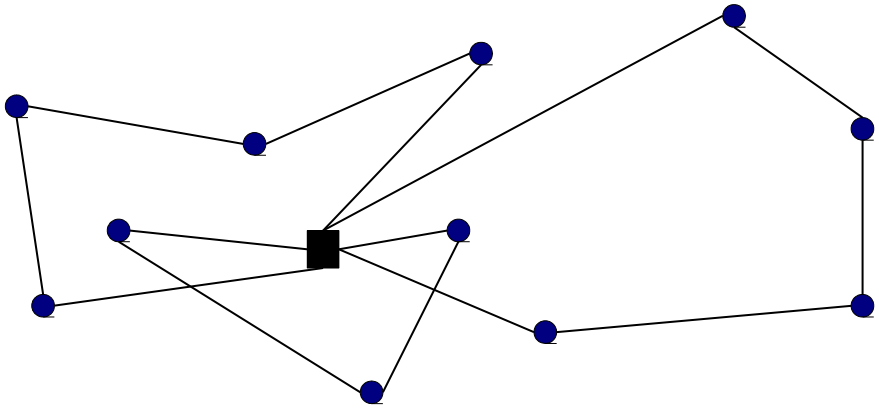

Figure 28: Tour Depletion Step 1

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

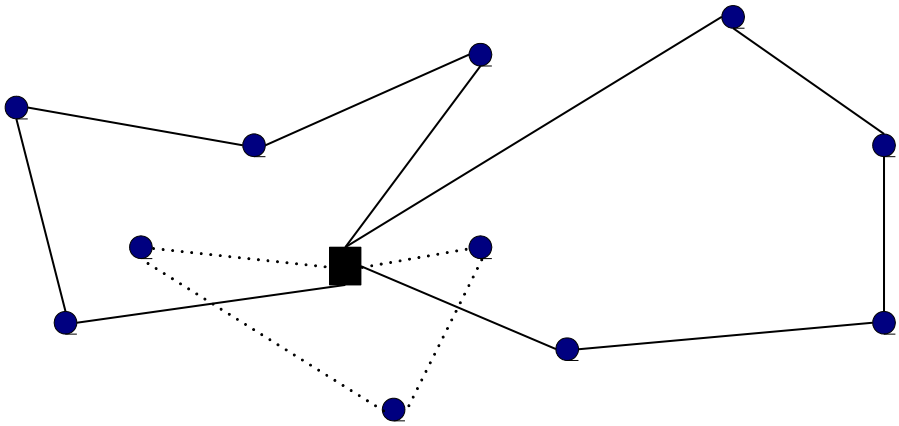

Figure 29: Tour Depletion Step 2

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

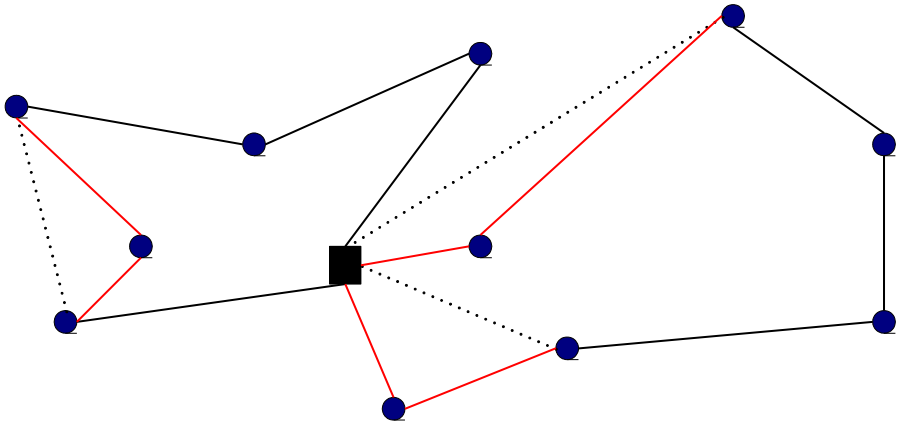

Figure 30: Tour Depletion Step 3

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.

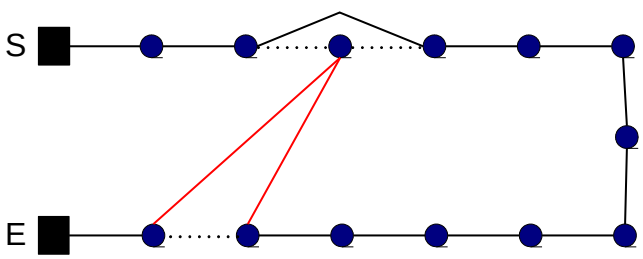

Figure 31: Relocate on same route

Or relocate a stop from one route to another.

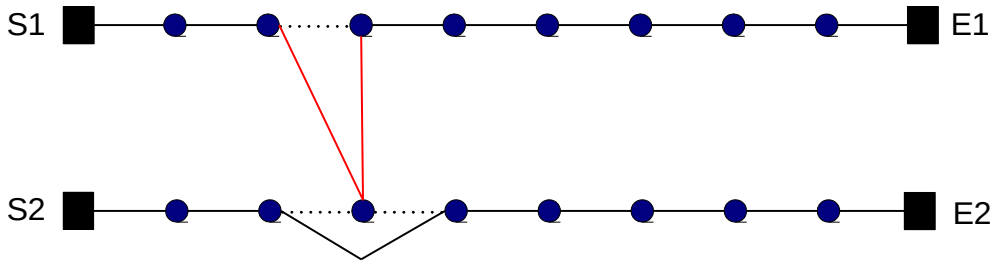

Figure 32: Relocate between routes

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.

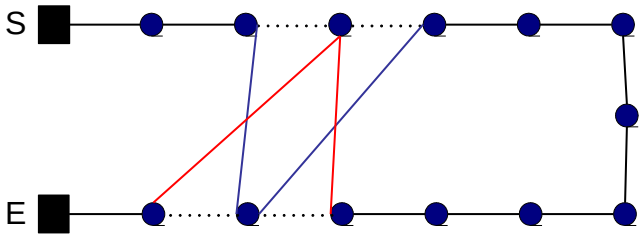

Figure 33: Exchange on single route

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

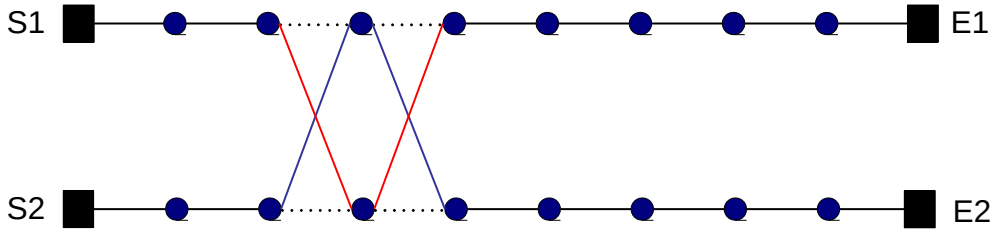

Figure 34: Exchange between routes

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.

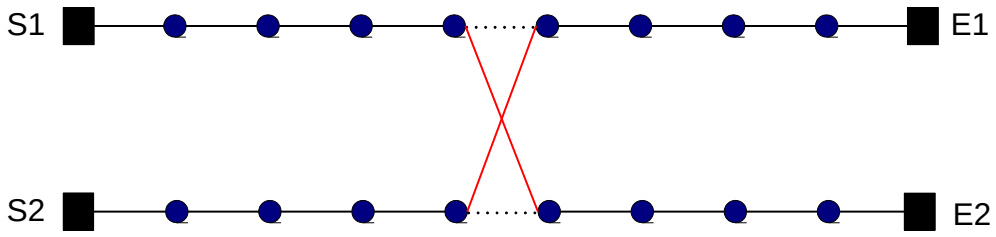

Figure 35: Cross operation

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.

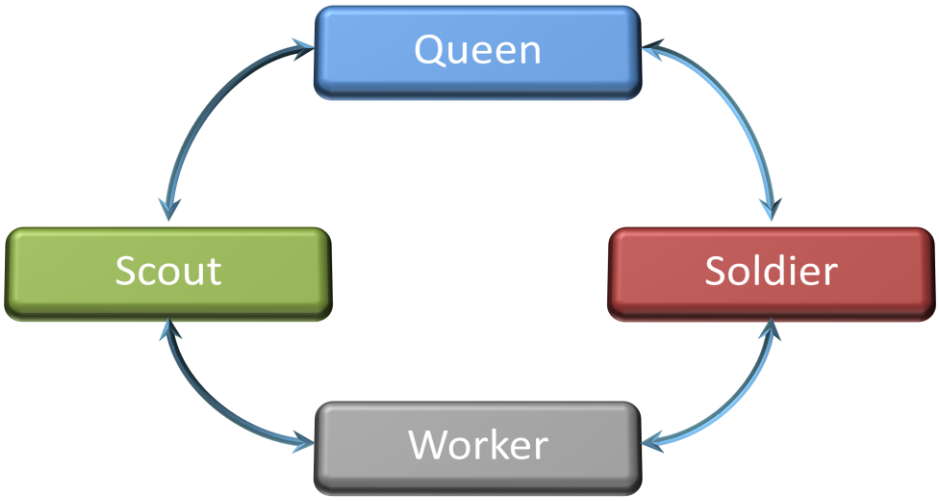

Figure 36: Ant type relations

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.

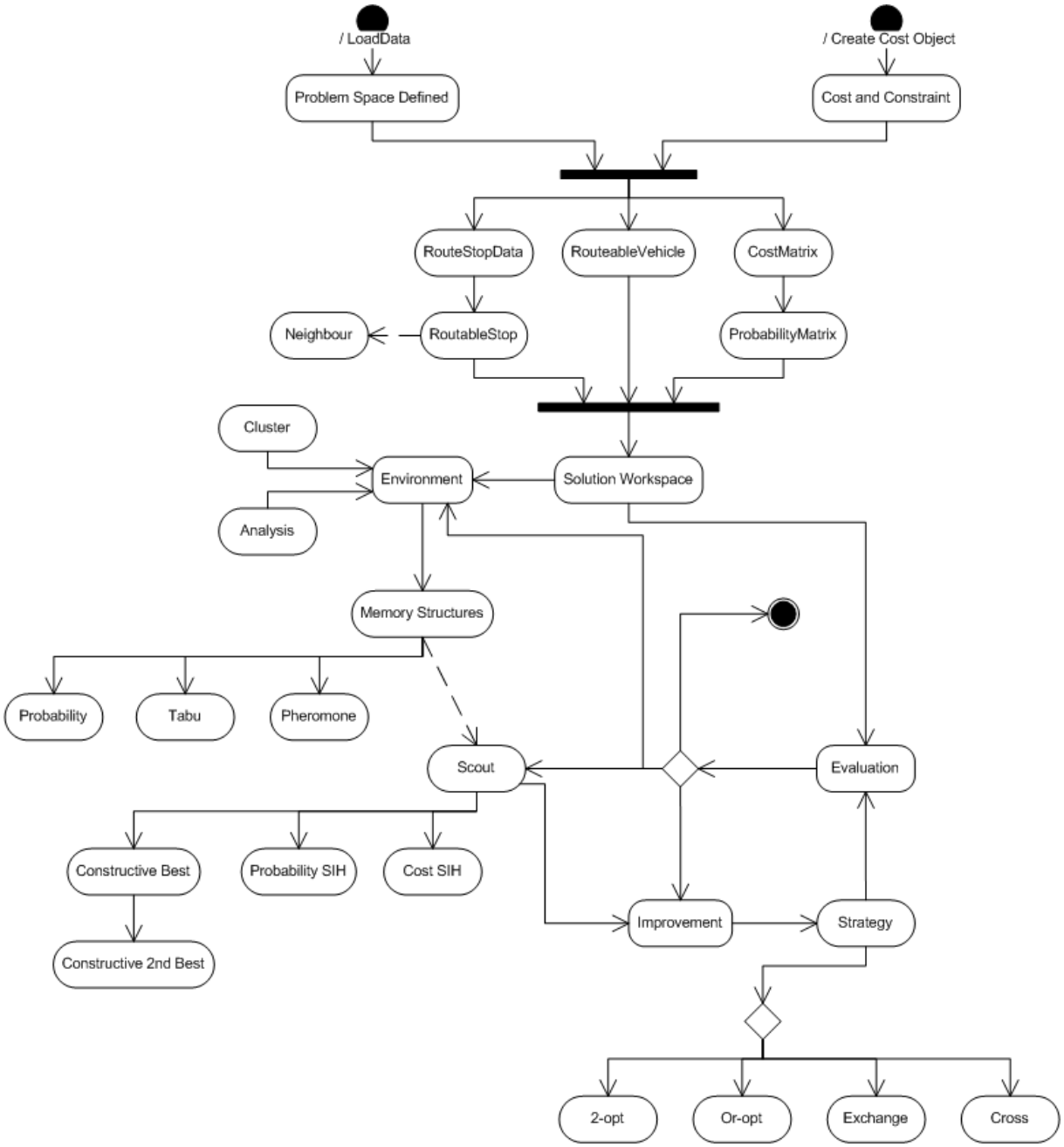

Figure 37: Ant System on Adaptive Objects - ASAO

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.