*Chapter 3*

# 3 PROBLEM SOLVING METHODOLOGY

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 their needs to be optimised.

Heuristic methods search only part of the solution space. This result in the quicker termination 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.

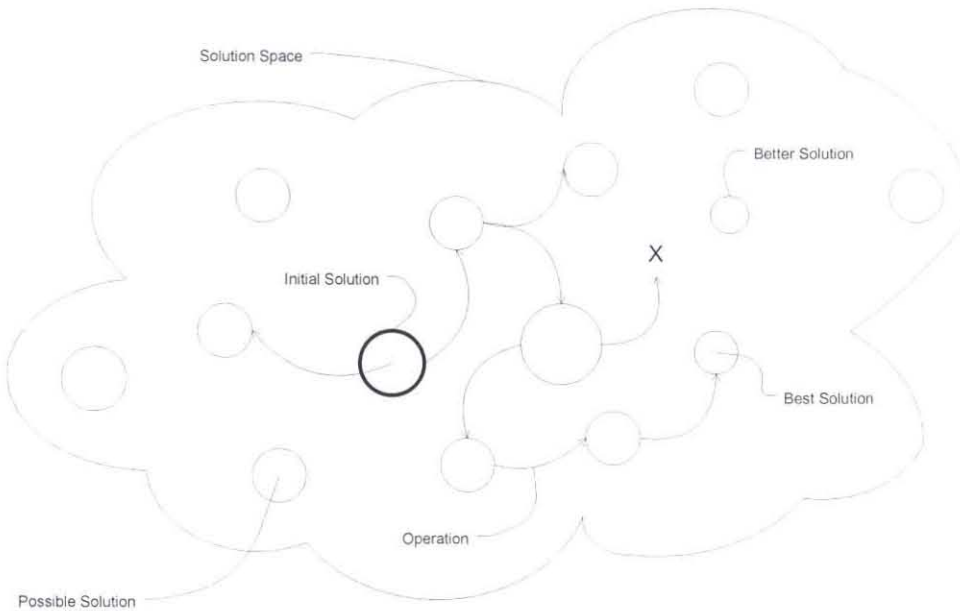


**Figure 10: Solution Space**

Figure 10 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.

A circle which size reflects the total cost of the solution represents a solution. The smaller the circle, the better the solution. This indicates that there are possible solutions that is not cost-effective and which we do not want to consider as an end result.

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).

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.

Our evolutionary metaheuristic 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.

In the first stage we generate an initial solution with the help of a construction heuristic, in this case we make use of the sequential insertion heuristic (SIH). This method results in a solution that is feasible but not necessarily the best. The

feasibility of the solution ensures that it existing our solution space (see the initial solution in Figure 10).

The improvement stage traverse from our current position to a neighbour's solution. Because solutions do not truly exist in our environment, we need to generate a new feasible solution. This is done by applying an operation on the current solution. As we progress it can happen without an already existing solution is generated by an operation. This can result in cycles in our search path, which leads to revisiting existing solutions and result in unnecessary computational time. One of our objectives will be to prevent such cycling. After a specified number of iterations we have 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 the figure we can see that the path to the best solution might have to go through a not so good 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 is implemented with the 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 following sections will discuss in more detail the specific methods used to traverse through the solution space. It will also point out where knowledge about

the problem beforehand can have an effect on the implementation of the solution. The sections consist of the problem representation in objects, the approach of the solution, a discussion on the construction heuristic and improvement heuristic.

## 3.1. Objects.

In the previous chapter we presented the problem in a mathematical model. This model has the purpose of describing the parameters of the problem as well as the conditions it has to meet. Implementing a solution for the problem is not as easy as describing it. This section will explain the components we utilise for solving the problem. The solution was designed in an object orientated way.

The object model is divided into two areas. Model will describe the problem objects or the input data. The second model will describe the alterations on the problem objects and the additional objects required to produce a solution. An object consists of properties, methods and relations.

### *3.1.1. Problem objects.*

This section will discuss the mapping from the input data to the objects in the solution. We need to identify all the objects represented in the input data. Let us consider the vehicle routing problem again.

The basic VRP consist mainly of a depot, stops and vehicles. A depot can be seen as a specific stop with certain properties.

*Stops*

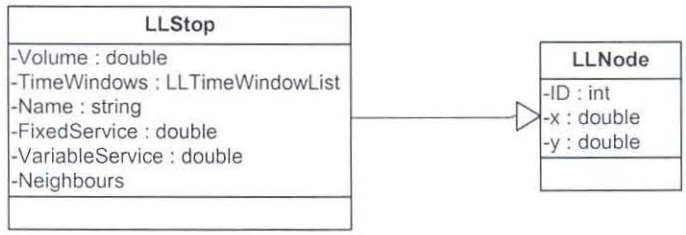


**Figure 11: Problem object Stop**

The above object represents a stop. A stop must comply with the basic functionality of a graph node. The figure indicates that a stop inherits all the properties and methods of a node. The properties of a stop is as follows:

- ID - a unique value to identify the stop.
- X, Y - the spatial representation of the node
- Volume - the volume that a stop will utilise on a vehicle
- Time Windows – a list of available time windows that a stop can be visited in.
- Name – a descriptive name for the stop for display and report purposes
- Fixed Service – the fixed service time for a stop in minutes. This represents the stopping time required at a stop without loading or unloading anything.
- Variable Service – this represents the volume per minute rate of loading or unloading goods at the stop. The total service time at the stop consist of the fixed service time + (volume * variable service time)

- Neighbours – this is a list of neighbours that a vehicle can visit from a stop. In the basic VRP this list will consist of all the other stops. In our problem that includes time windows, it might happen that it will never be feasible for a vehicle to travel from one stop to another because of time window compatibility (see description of time window compatibility), which basically means that the following stop has time windows that ends before the current stop's time windows begin. Science has shown that we cannot travel back in time and thus we will not consider this stop as a neighbour.

Operations required by the problem model for stops can be defined as followed:

- Travel Time - working with the restriction of time widows, we need to know that time it will take to travel from one stops to another to ensure that we arrive at a feasible time. We implement travel time between stops in a matrix. One of the additional constraints to our problem is the requirement to calculate the travel time depending on the time of the day. The travel time function accepts the two stops in the travelling sequence and the time of departure from the first stop. See this section on the cost matrix for further detail.

- Distance - distance is calculated in a similar way as travel time. Distance is also dependent on time of day because the travel time between two stops determines the route between the stops. This means basically that a quicker route might not be the shortest.

### *Depot*

The properties and methods of a depot is exactly the same as for a stop. In defining a depot, we define a single stop. Travel time and distance calculations applied on the depot in the same manner as for a stop.

Our solution considers only one depot, which implies that all the existing vehicles and stops belong to that depot. Extending this problem to a multi- depot problem would require the depot object to be reconstructed by adding a stop list as well as vehicle list to the depot object.

### *Vehicle*

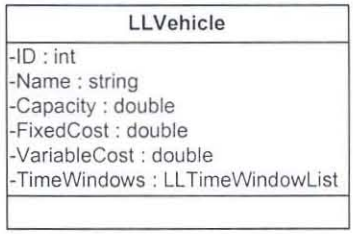


**Figure 12: Problem object Vehicle**

The vehicle object in our implementation consist of the following properties:

- ID - a unique key for identifying the vehicle
- Name - a descriptive name for display and reporting purposes
- Capacity - the total volume that a vehicle is capable to handle
- Fixed Cost - the cost of utilising this vehicle without even travelling
- Variable Cost - the running cost of the vehicle. Part of the cost of the route is calculated by Fixed Cost + (Variable Cost * Distance).

- Time Windows - a list of available time widows that the vehicle can be utilised.

There does not exist specific operations for a vehicle in the problem object model.

### *Time Windows*

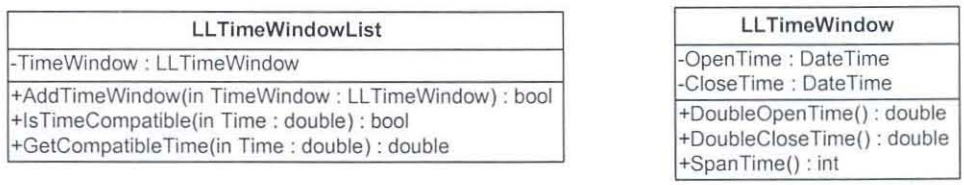


**Figure 13: Problem object Time Window**

Time widows play an important role in the problem. All of the problem objects, namely depot, stops and vehicles, are associated with a time window list to indicate availability for the object's specific function.

Time window consist basically of an open and close time. This time is saved in a datetime format to allow for implementing problems that span across multiple days. Operations on the time window includes:

- DoubleOpenTime - returns the number of minutes after specific date time from a fixed time. This is done to allow the algorithm to work in a linear reference environment. Let us for example say that the open time is 07:00 on today's date. Calculating the linear time consist of the difference between the open time and today at midnight, which results in 7 hours. Converting the hours to minute's results in a linear open time of 7 * 60 = 420. If the open time was specified as yesterday at 07:00 the difference between today at midnight and the open time is –17 hours. Converting

the hours to minute's results in a linear open time of -17 * 60 = -1020. Although the value is negative is still valid for a linear scale.

- DoubleCloseTime - returns the number of minutes after a specific date time, same as DoubleOpenTime.

- SpanTime - returns the difference between the open and close time in minutes.

What we can see from the time window properties is that our linear timescale consists of minutes. The fixed point on the scale to calculate the linear values from is today's date.

The time window list object consists of a list of time windows. Operations on this list include:

- IsTimeCompatible - this function accepts a time and determines if there exists a time window that include the time, i.e. the time is after the open time and before the close time for a specific time window in the list.

- GetCompatibleTime – this function accepts a time and calculates the earliest available time according to the time window list. If no such time exists, an exception is thrown, which indicates incompatible time.

### *3.1.2. Solution Objects*

This section will give an overview of the solution objects used in the algorithm. It is important to understand this basic building blocks in order to see how the algorithm functions. Solution objects consist of extensions of problem objects to handle new information required by the solution, as well as help objects that play a role in solving the problem.

***Route Vehicle***

The implemented solution focus on deterministic data, i.e. all the demands and vehicles are available and known before the start of the solution. In terms of the vehicles the algorithm will not propose a best-suited fleet from a set of vehicles, but accept the vehicles as existing and ready to use according to their specifications. It can be simplified by allocating a route to a vehicle before we even start. The solution is therefore made up of a set of vehicles that contain routes.

One of the additional requirements of the problem is to allow for multiple routes on a vehicle. A vehicle can thus have multiple routes.

A vehicle with routes will be the main output of the system. A route vehicle is the input vehicle with routes associated to it.

***Routes***

A route can be seen as a sequence of stops that is visited by a particular vehicle at a specific time.

1.1.2.3 Route Stops

The determining of a best solution relies mainly on the handling of the stops. | A route stop consist of a stop with additional info such as:

- Arrival Time – the time a vehicle arrives at a stop
- Wait Time – the time a vehicle must wait at a stop before it can start servicing the stop.
- Service Time – as specified by the stop service time.

- Departure Time – the time the vehicle leave a stop for its next stop. This must be equals to the Arrival Time + Wait Time + Service Time.

- Next Stop – An indication on where to go next in the route. This method is the principle method of providing information on the route. Adding or deleting a stop from a route is made easy by just replacing the next stop. Adding a new stop requires replacing the current stop's next stop with the new stop and the new stop's next stop to the current stop's next stop. Deleting is as easy as setting the previous stop's next stop value to the current stop's next stop value. This only indicates the method of inserting and deleting a stop from a route and not the validity of the move.

### *VRP Base*

The main purpose is to solve the VRP. There exist several ways to solving a VRP. This object is the base object for the solution. The object contains all the necessary data and manipulates all the necessary methods applied on the data. The end result of the algorithm is the VRP object, which contains multiple solutions.

### *Cost*

Cost is defined as the cost in terms of distance and travel time from one stop to another. A cost matrix is used for storing the values.

The solution implements a cost function with time windows to represent the difference of cost on a link depending on the time. This basically result in a cost function that is a function of the time of day. When the algorithm requests a travel time from the cost function, the function first determines the cost matrix to use. This is done by finding a cost matrix, which time windows will contain the time provided. The cost for that time is returned.

It is important to notice the influence of such time dependent cost function in the solution. The advantage is that a more accurate route can now be constructed, which is very important for the success of the algorithm. When a vehicle travels from point A to point B, it will definitely take him longer during traffic peak periods. The use of an average travel time on a link will no be sufficient to take care of this problem. When a vehicle travels during peak time, his actual arrival time at the customer will be later than planned. Although the vehicle might make up this time during the off-peak time, the use of multiple time windows can result in a lateness that fall between two time windows, which result in additional wait time, which makes it more difficult to make up during the off-peak times.

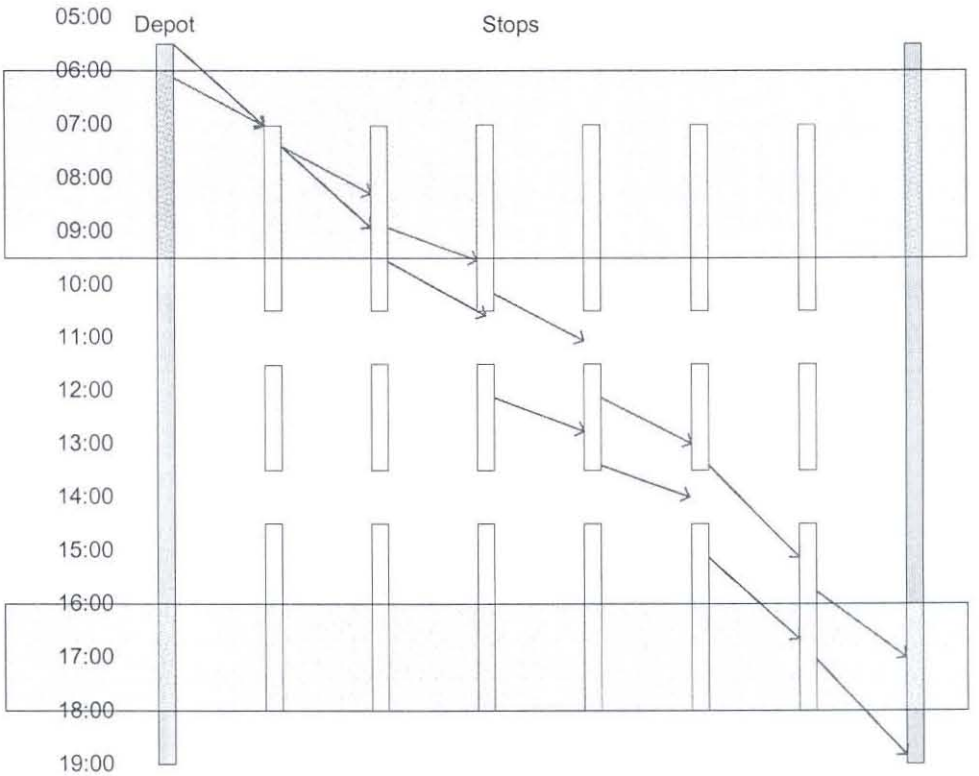


**Figure 14: Peak and Off-Peak travel time influence**

Figure 14 explains the importance of a time dependent cost function in the solving of the VRP. The figure represents a typically delivery day with stops that has similar time windows. The patterned areas represent peak traffic time. A route is constructed from the depot on the left back to the depot on the right.

The green arrow line represents the route making use of an average travel time on a link. The red line represents the actual travel time. Starting of, we can immediately see that the average route departs later that the actual route. This is because the departure time from the depot is determined by the open time of the first stop. The slope of the red line is steeper than the green one, which indicates a longer time to travel from the depot to the first stop in the actual route.

The algorithm will ensure that the arrival time at the first stop is as early as possible. In the above case, both routes arrive at the open time of the first stop. The service time is not affected by the cost function and both routes depart from the first stop at the same time.

During the peak travel time, the actual route requires a bit more time to travel than the average travel time. At stop 3, the actual time arrives too late to be serviced in the first time window and has to wait for the second time window to take effect. Although the actual travel time is quicker than the average time during off-peak periods, the aggregated loss due to lateness cannot be recovered. This is mainly due to the synchronisation of the stop time windows.

The example above is proof that we need to implement a time dependent cost function in the algorithm to produce more realistic results.

The VRP is a NP-hard problem, which suggest that it is difficult to solve. Heuristic methods can provide feasible solutions in reasonable time, but additional constraints will increase computational time. The addition of a time

dependent cost function requires the algorithm to recalculate the travel time between two stops every time a new stop is added to the route or a stop is removed from the route. This is necessary for all stops after the added or removed stop, as the addition of a stop will alter the arrival time of all subsequent stops.

### *Solution*

A solution object represents a possible solution to the VRP problem. The solution contains route vehicles and their corresponding routes and stops, as well as an orphan list of stops. A solution object is used to generate more solutions from through an operation.

Although the algorithm considers all the main influential parameters, we cannot ignore the human factor. There might still exist a preference from the user regarding a specific solution. During the execution of the algorithm the proposed methodology requires a list of solutions to be able to traverse through the solutions space. We propose that the algorithm does not only present the user with the best to solution found, but provide the option of selecting one of the best solutions. Practical implementation has shown that the best calculated solution might not always be the most feasible for the client. This might be because of the customer driver relationships, driver knowledge of areas, etc.

### *Construction Heuristic*

The proposed solution requires some possible solution to start working from. There exist multiple methods of constructing an initial solution. In a later section the selected construction heuristic namely the Sequential Insertion Heuristic (SIH) will be discussed. The algorithm can function from an existing solution. In those cases, the construction heuristic would not be necessary.

Working in the ASP environment implies dynamic acquisition of data from clients. The solution has to take into consideration the possible extension of the current implementation, i.e. there might exist a better construction heuristic for the specified problem. For that specific reason we propose the implementation of a construction heuristic in the main algorithm. This will allow the addition of other construction heuristics in the future. The current construction heuristic already produces multiple solutions for the improvement heuristic to work on.

***Improvement Heuristic***

The implementation of an improvement heuristic is the focus area of this research topic. The VRP object contains an Improvement Heuristic method. As in the case of the construction heuristic, the VRP is force the existence of such a method, but does not determine the implementation detail.

### *3.1.3. Problem Helper Methods*

This section will discuss the systematic approach in solving the problem. Although the focus of this thesis is on designing a new VRP solution, we cannot ignore the implementation environment. The ASP environment has a major influence on the line and implementation of the solution algorithm. The main reason is because of the unpredictability of the data.

The next paragraphs will discuss information flow and manipulation through the process.

***Input Data and Object Generation***

The first step towards a feasible solution is to acquire data from the client. There exist multiple methods of transferring data from the client information service to the ASP server. This is the topic of another study.

What is important is that the data must be complete. This means that the incoming data must contain all the necessary information. In addition, we must know where the incoming data is headed for, e.g. the client must specify which value from a stop is the demand and which is the time windows etc.

The client data must now be constructed in the defined objects. The algorithm requires data that is relevant to one depot and one instance of a routing schedule. This means that a stop will only be visited once during the time windows specified.

After this step, the algorithm will contain all the necessary data.

### *Solution methods*

As explained in previous sections, a route consists of a sequence of stops. The manner in which the structure is maintained is important in the manipulation procedures of the algorithm. This paragraph describes basic actions allowed on a solution. The implementation of the construction and improvement heuristics will depend on the stability of these actions.

*Route stop addition*

As mentioned previously in the discussion of the time dependent cost function, the addition of a stop on a route has several consequences on the subsequent stops.

The addition of a stop in a route results in this shift of the arrival time of subsequent stops, which can result in time window incompatibility, i.e. the arrival time is not sufficient anymore to be able to serve the stop in its available time windows. An action of inserting a stop in a route that result in incompatible time windows must flag the route as invalid.

The removal of a stop on a route has less dramatic results, i.e. if a route was valid before the removal of a stop, it can still be valid. It might not be as efficient, but it will still exist in the solution space.

The addition of a stop on a route also has an effect on the vehicle volume. Adding a stop increase the volume required on the vehicle. The addition of a stop can result in a route that exceeds the vehicle capacity. This action must flag the route as invalid.

The removal of a stop result in the decrease in the required volume for the vehicle. The removal of a stop from a route cannot result in a vehicle that exceeds capacity.

It is important to know that the weight and arrival time calculations have to be executed on each insertion and removal of a stop in a route. The implementation of these methods must be effective.

*Vehicle stop addition.*

The addition of a stop on a route has an effect on the overall routes associated with the vehicle.

When a stop is added on a route, the route's departure and arrival time from the depot change. This can result in a delay in the departure of a next route from the depot. The new departure time for the next route can result in incompatible time windows at stops, or even an incompatible time window for the route vehicle. The addition of a stop on a route can result in the invalidity of subsequent routes and the route must be flagged accordingly.

*Time Window Compatibility*

The concept of a time window compatibility matrix as proposed by van Schalkwyk, [52] has not been proven, but has a logic sense to it. The calculation

of such a matrix can be done at the beginning of the algorithm, which adds to the setup time, but not the running time.

An aspect not catered for in the proposal of the TWCM is the variation in the travel time depending on the time of the day. The addition of variable travel time adds some complexity to the problem. In Figure 15 we show effect of the variable travel time.

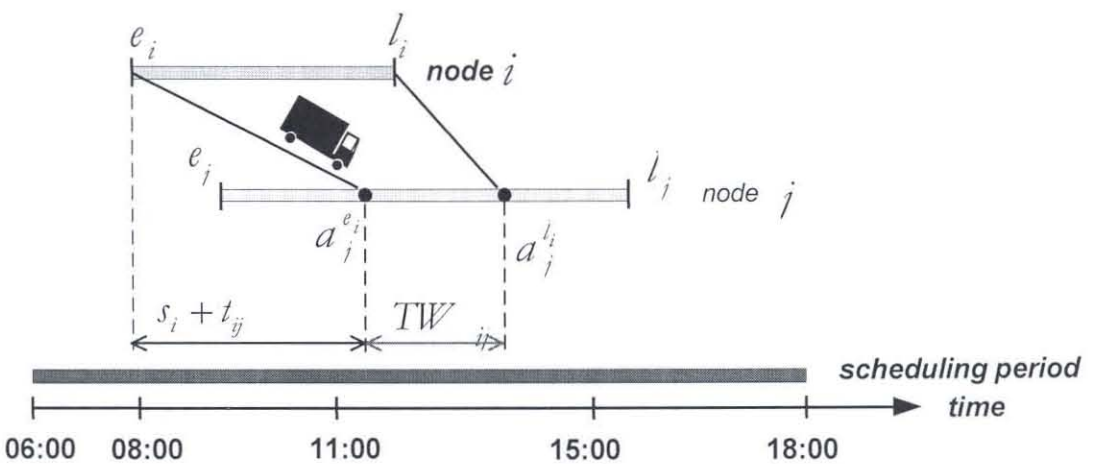


**Figure 15: Variable Travel Time on Time Window Compatibility**

From the figure we can depict the effect of the variable travel time. In this implementation, the travel time function is not a continuous function, but a disjunctive function consisting of constant times at specific intervals. In our calculation of the TWC, we need to overlay the travel time function's time windows with that of the source stop. We determine travel time from the source stop's departure time.

## 3.2. Approach

The approach consists of different phases, which will be discussed in more detail in the following paragraphs. The first phase consist of the generation of the required distance and time matrices for specific time periods. The second phase is the generation of an initial solution through a construction heuristic. This is necessary for the improvement heuristic that follows. The improvement heuristic will follow the guidelines of the Tabu Search. The heuristic will search for a good solution by diversifying and intensifying the solution area. After a predetermined number of iterations, or if a termination parameter is met, the post optimization phase will ensure that the current best solution is optimised to its local minimum.

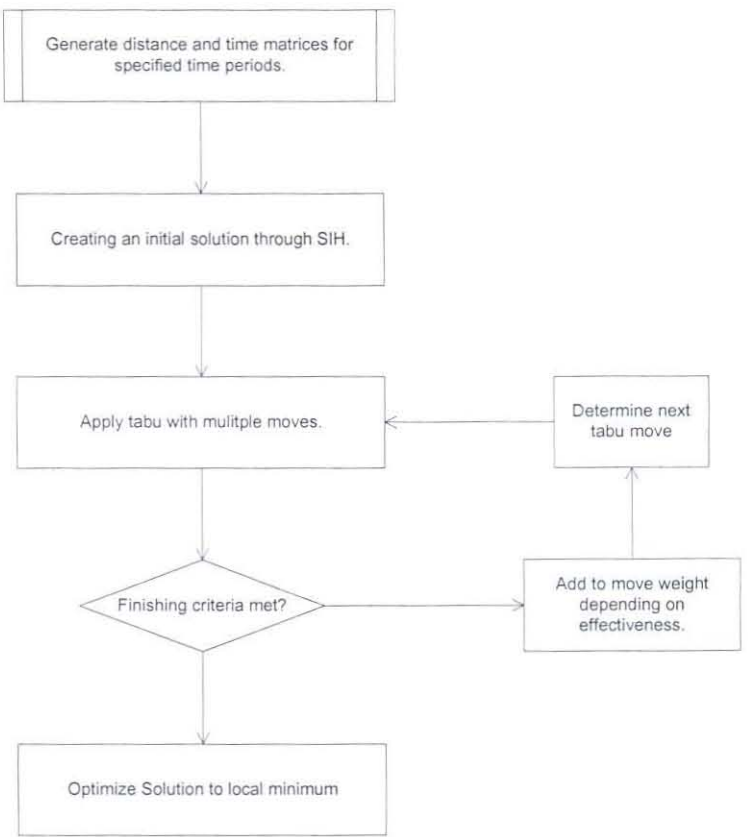


**Figure 16: Algorithm Phases**

## 3.3. Initial Solution

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

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 and 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, unrouted 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. The parameters $\alpha_1, \alpha_2, \mu$ and $\lambda$ are used to guide the heuristic.

Inserting customer $u$ between $i$ and $j$ increases the length of the route by the distance insertion, $d_{iu} + d_{uj} - m\,d_{ij}$. 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 unrouted 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 time window compatibility 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 TWC, 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.

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) [21]. 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 $e_0$, 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.

We also extend the Push Backward to incorporate the vehicle time windows. Inserting a customer u as the first stop in the route advances the departure time at the depot depending on the open time of the depot, the best available time of the vehicle and the open time of the customer $u$. The vehicle would leave the depot at $\max\{b_i=0, b_k, b_j - t_{ij}\}$ where $b_i=0$ is the open time of the depot, $b_k$ the open time of the vehicle and $b_j - t_{ij}$ the open time of u retracting the travel time from $i$ to $j$.

### 3.4. Improvement Heuristic

Chapter 2 discussed heuristic techniques we considered for implementing a solution for the specified VRP problem. It suggested the use of a meta-heuristic technique. Meta-heuristics use information of the problem environment and the nature of the objective function to direct the search process to regions that promise better solutions.

Although there exist many alternatives in selecting the appropriate tool, the success of these methods depends on many factors, like their ease of implementation, their ability to consider specific constraints that arise in practical applications and the high quality of solutions they produced.

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 TS 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 TS 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 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 in put data, we implement new operations and add some statistical selection method in the guidance algorithm.

### *3.4.1. Operations*

#### *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 test the validity of the insertion of the orphan stop after the neighbour stop. The move is accepted if the insertion is valid.

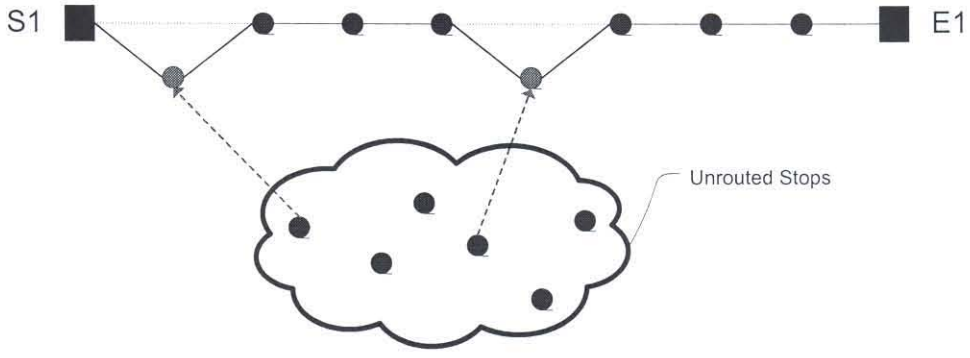


**Figure 17: Insert Operation**

#### *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.

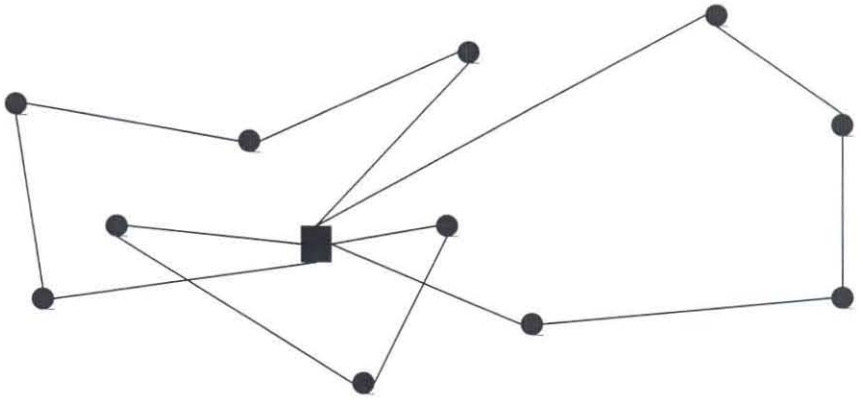

**Figure 18: Tour Depletion Step 1**

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

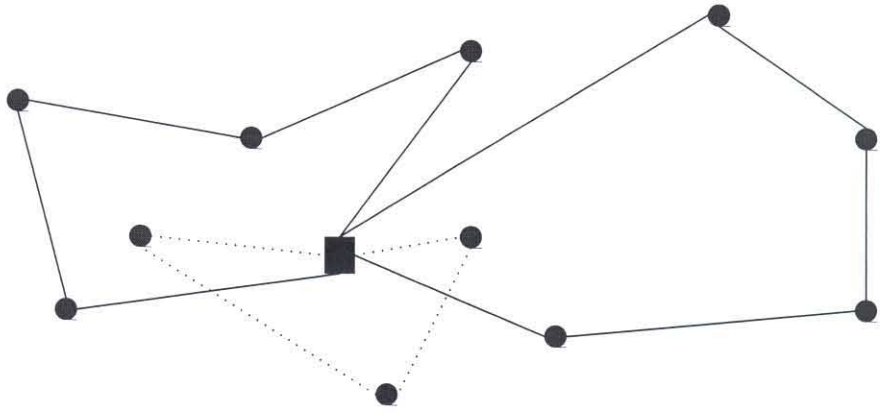

**Figure 19: Tour Depletion Step 2**

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

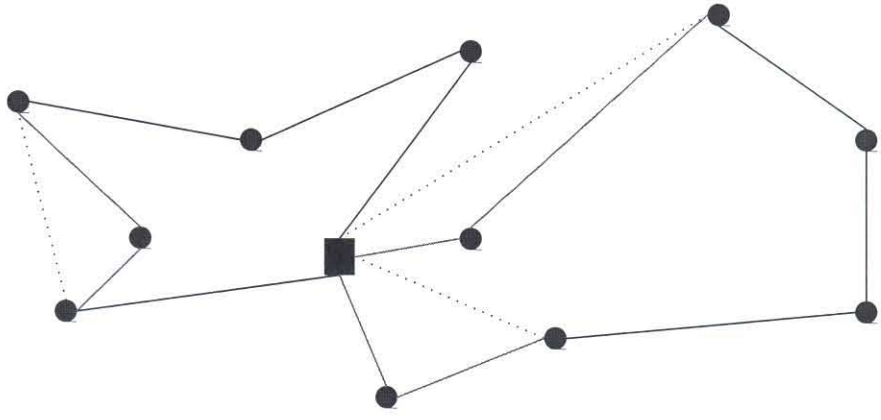

**Figure 20: Tour Depletion Step 3**

An additional criteria 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.

### *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.

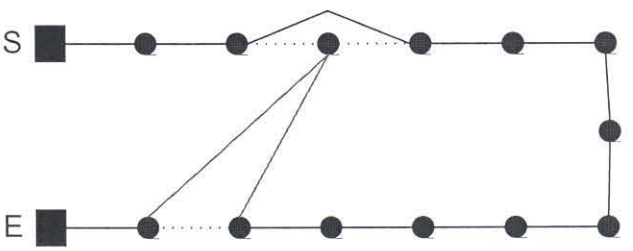


**Figure 21: Relocate on same route**

Or relocate a stop from one route to another.

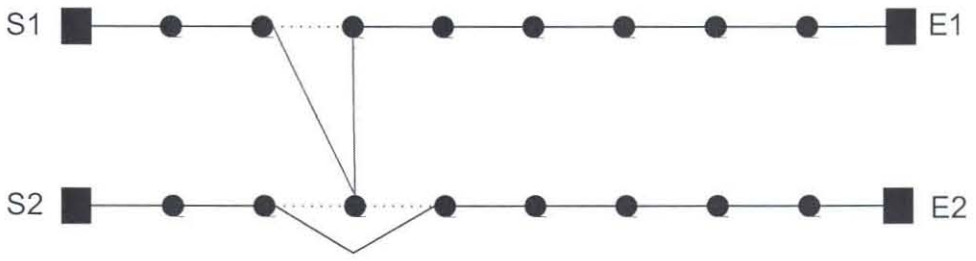


**Figure 22: Relocate between routes**

### *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.

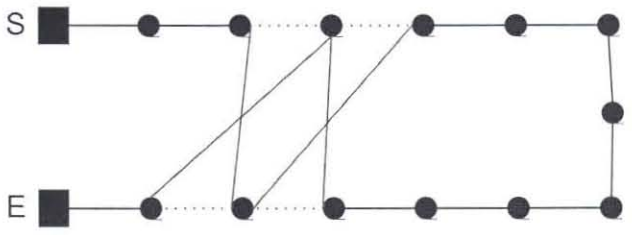


**Figure 23: Exchange on single route**

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

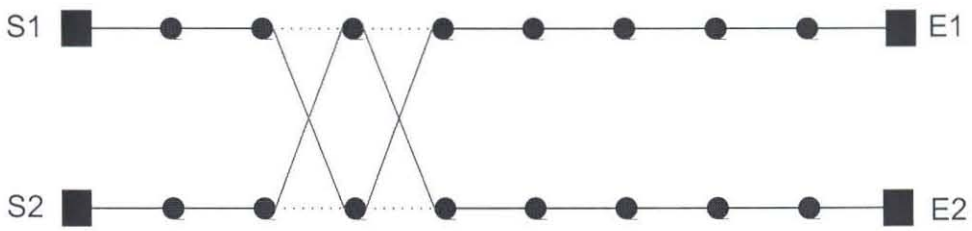


**Figure 24: Exchange between routes**

***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.

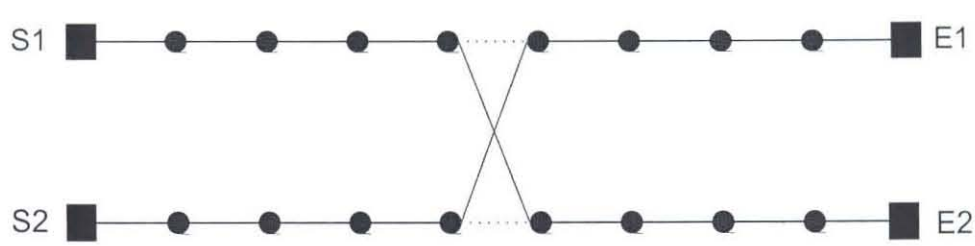


**Figure 25: Cross operation**

***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 exist 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.

***Double Fit***

The operation tries to fit routes or segments of routes as additional routes on a vehicle. This action will result in the use of fewer vehicles.

The double fit operator has the purpose of filling up a vehicle to its time window capacity. The operator will test form time available on a vehicle and if there exist a continuous time that is greater than a minimum time specified, the operator can look for stops that fit in that time frame. If a route can be constructed to fill the open time slot, the move is accepted and results in other routes that have fewer stops. This move can now result in probable tour depletion after some optimization on the routes.

### *3.4.2. Guidance Algorithm*

Meta-heuristics use information of the problem environment and the nature of the objective function to direct the search process to regions that promise better solutions. The implementation of the guidance algorithm has an important effect on the effectiveness of the algorithm.

The implementation of the guidance algorithm utilise aspects from different sources. A simulated annealing approach is followed in an oscillating fashion.

Neighbourhood search methods are also selected randomly in a statistical learning fashion. Each operation has its own tabulist.

***Statistical Selection***

The implementation of all the specified operations can lead to inefficient computational time utilisation. Depending on the manner if the input data, some operations can be more effective than other, or can be ineffective in situations.

When the input data has customers with tight time windows, the capacity of the vehicle does not really play an important role in the solution, as the vehicle does not have time to visit enough stops to load the vehicle to capacity. The double fit operation will not be effective on these types of data. The statistical selection will eliminate the use of this operation.

The idea of the statistical selection is to determine the success rate of an operation. When we randomly select an operation, the probability of the selection of a specific operation depends on the success rate. When we start the improvement heuristic, we assign an equal value to the success rate of all the operations in the list. On the first iteration, the probability for an operation to be selected is the same for all. If the operation completes successful, we increase the success rate by a value depending on the type of success. This increase will not have a major effect in the beginning, but after a number of iterations, the more successful operation's success rate will increase, and that will increase the probability of the selection.

***Simulated Annealing***

Another control mechanism implemented by the guidance algorithm is derived from the simulated annealing procedure. 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

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

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:

$$P(accepting\ a\ move) = e^{(-\Delta/T_k)} \quad (2)$$

This is a modified version of the annealing equation, where $\Delta$ = 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

$$T_r = \max(T_s/2, T_b) \quad (3)$$

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. The Tabu Search is used to search the local neighbourhood.

## 3.5. Conclusions

This chapter describe the design of a solution algorithm that is capable to solve the VRP in an ASP environment. The additional constraints imposed by the ASP environment are incorporated in the design of the algorithm.

The problem is partially solved by the introduction of new operations on the solution as well as extensions of current existing operations. The guidance algorithm implements multiple operations, which allows it to be effective on all types of input data. The statistical selection of operations is believed to improve the effectiveness of the algorithm.