Chapter 3

Intelligence in solution algorithms

Once a model of the perceived reality is formulated, a process is required to obtain a solution to the model which, in turn, could be implemented to solve the problem in reality. It should always be noted that *models* are solved, not *real problems*. Rardin (1998) refers to *numerical search* as the process of systematically trying different choices for decision variables so that the best feasible solution could be found. This chapter is dedicated to review the three primary search strategies used to solve mathematical programming models.

The first of these are exact solution algorithms where one can prove that the *best* feasible solution found is in fact the global optimum for the problem. The first section of the chapter introduces some of the fundamental exact solution algorithms, with reference to further review articles for interested readers.

Exact solution algorithms are unfortunately not always viable when the size of a problem increases. To compensate for the time-consuming computational burden, solution seekers opt for *approximate* solutions, also referred to as *heuristics*, where the *best* solution may, or may not, be the true optimum for the problem. Yet, heuristics offer solutions that are often better than the typical industrial solutions obtained through intuition and common sense. The second section introduces a number of heuristics dating back from the 1950's, and follows a few variations of these heuristics.

Heuristics have evolved during the 1990's to what is referred to as *metaheuristics* — intelligent strategies governing the execution of various heuristics in order to find even better solutions. The third section introduces a number of metaheuristics and its variations, from where a conclusion is drawn and a motivation is provided for the choice of solution algorithms for this thesis.

3.1 Exact solution algorithms

It might at first seem counterintuitive that integer (discrete) linear and combinatorial problems are more difficult to solve than their continuous counterparts, seeing that the algebra for Linear Programming Problem (LP) algorithms can be quite daunting. A discrete model with a finite number of possible decision variable values, on the other hand, seems much easier. This reasoning holds only for small instances of discrete problems, and Rardin (1998) confirms that total enumeration of all possible combinations is the most effective method to find the best solution. Consider a problem with only two binary variables, x_1 and x_2. There are only $2^2 = 4$ possible cases. Although ten binary variables will require $2^{10} = 1024$ cases to be enumerated, it is still viable using computers. The exponential growth in the number of case evaluations when enumerating requires alternative algorithms for problems of practical size.

This thesis follows the classification proposed by Laporte and Nobert (1987) and Laporte (1992) whereby exact algorithms are grouped into three primary categories, each covered in the following subsections.

3.1.1 Direct tree search methods

The analogy of a tree in search methods represents the primary stem being some initial solution, from where the stem is split into branches, or secondary stems that are related to the primary stem. These secondary stems, in turn, branch into tertiary stems, etc.

The first step in direct tree search methods is to find the primary, or initial solution. Because discrete optimization models are typically hard to solve, it is natural to find related, yet easier formulations of the problem. Auxiliary models are referred to as *relaxations* of the original discrete problem and are *easier* to solve as some of the constraints, or objective function(s) of the discrete problem are weakened. Solving the relaxations can lead the modeler to make solution interpretations of the original problems. Various relaxation techniques vary in *strength*. Rardin (1998) defines a relaxation as *strong* or *sharp* if the relaxation's optimal value closely bounds that of the original model, and the relaxation's solution closely approximates an optimum in the original problem. Various relaxation methods exist in introductory Operations Research textbooks and include LP relaxations, stronger Big-M constants and the introduction of valid inequalities (for example the *Cutting Plane* algorithm (Jeroslow, 1979)) (Hillier and Lieberman, 2005; Rardin, 1998; Taha, 2003; Winston and Venkataramanan, 2003). An even stronger relaxation, referred to as *Lagrangian* *relaxation*, do not weaken integrality constraints, but rather relax some of the main linear constraints after which they are dualised (or weighed) in the objective function with Lagrangian multipliers (Fisher and Jaikumar, 1981). Desrosiers et al. (1988) use Lagrangian relaxation methods to solve a variant of the MTSP with time windows.

Once a relaxation is solved to optimality, and the solution also conforms to all constraints in the original problem, the solution *is also* the optimal solution for the original problem. If not, various strategies and algorithms are employed to systematically work towards a relaxation of which the optimal solution is also optimal for the original problem.

The *branch-and-bound* search algorithms combine relaxations with an enumeration strategy to find optimal candidate solutions, while bounding the search by previous solutions. Laporte et al. (1989) adapt the branch-and-bound algorithm in solving a class of stochastic location routing problems with networks of 20 and 30 nodes. Laporte et al. (1986) solve the asymmetrical Capacitated Vehicle Routing Problem (CVRP) for 260 nodes. The structure of the VRP and its relationship with one of its relaxations, the MTSP, is exploited by Laporte (1992) in a similar manner.

The branch-and-bound algorithm has been modified with the introduction of stronger relaxations prior to the branching of a partial solution. The modified algorithm is referred to as *branch-and-cut* as the stronger relaxations are obtained with the inclusion of new inequalities. The inequalities should hold for all feasible solutions of the original discrete problem, but should render the last relaxation's optimum as infeasible, hence the term *cut*. Padberg and Rinaldi (1987) illustrate the generation of cuts in a symmetrical TSP with 532 nodes. Laporte et al. (1992) describe a general branch and cut algorithm for the VRP with stochastic travel times. The authors introduce cuts in the form of subtour elimination constraints, and introduce lower bounds on penalties if a route exceeds its predetermined route duration limit.

Van Slyke and Wets (1969) introduce the L-shaped method as a variant of the cutting plane algorithm for specific linear programs. Birge and Louveaux (1988) acknowledge that the method holds opportunity for stochastic programming applications and modify the method to create multiple cuts in each major iteration. Laporte and Louveaux (1993) further expand the method and refer to their general branch-and-cut procedure as the *integer L-shaped method* and apply it to stochastic integer programs and note that fathoming rules are different than in branch-and-bound trees. In another variation on branching, Christofides et al. (1979) propose a depth-first tree search in which single feasible routes are generated as and when required in their VRP formulation based on the TSP.

In their recent review of exact algorithms based on branch-and-bound, Toth and Vigo (2002a) state that these types of algorithms remain the state of the art with respect to exact solutions, especially in the case where asymmetric cost matrices exist. Giaglis et al. (2004) confirm that exact approaches are applicable to problems of practical size only if they have low complexity.

3.1.2 Dynamic Programming (DP)

DP determines the optimum solutions of an n-variable problem by decomposing it into n stages each consisting of a single-variable subproblem (Taha, 2003). The objective is to *divide-and-conquer* real-life problems by enumerating in an intelligent way through a state space of solutions (Brucker, 2004). In solving shortest path problems, Rardin (1998) claims that DP methods exploit the fact that it is sometimes easiest to solve one optimization problem by taking on an entire family of shortest path models. DP was first proposed for solving VRPs by Eilon et al. (as cited by Laporte (1992)).

Hamacher et al. (2000) faced the requirement that the nodes to be routed in a tour must be chosen from a small region of the map, and motivate their choice by the fact that the truck drivers have a local knowledge of the environment and is subjected to business constraints. Although the most natural DP formulation results in a DP with infinite state and action spaces, an optimality-invariance condition recently introduced by Lee et al. (2006) establishes leads to an equivalent problem with finite state and action spaces. Their formulation leads to a new exact algorithm for solving the Multi-Vehicle Routing Problem with Split Pick-ups (MVRPSP), based on a shortest path search algorithm, which they claim to be conceptually simple and easy to implement.

Although in a different problem context, Beaulieu and Gamache (2006) present an enumeration algorithm based on DP for optimally solving the fleet management problem in underground mines. Their problem consists of routing and scheduling bidirectional vehicles on a haulage network composed of one-lane bidirectional road segments.

Li et al. (2005) integrate the machine scheduling problem with a delivery routing problem and formulate a DP recursion since there are a finite number of time points for the start time of a trip of the vehicle. They conclude, however, that the problem can be simplified by limiting deliveries to direct shipments, a situation that is inappropriate if there is a large number of customers and small shipments across a geographically dispersed network.

3.1.3 Integer linear programming

The set partitioning formulation is a method that starts by assuming that the totality of routes which a single vehicle can operate feasibly, can be generated (Christofides et al., 1979). Let A denote the set of nodes representing customers. Then if $S \subseteq A$ is the subset of nodes which can be feasibly supplied on a single route by a vehicle v_k, it is assumed that the total variable cost associated with the optimal routing of nodes in S can be calculated. This is not trivial if $|S|$ is large as it relates to a TSP (one vehicle with a single route). For each vehicle v_k a family S_k of all feasible single routes for that specific vehicle is generated. A matrix $G = [g_{ij}]$ is produced with row i representing customer x_i and M blocks of columns where the k^{th} block of columns corresponds to vehicle v_k and column j_k of the block corresponds to a feasible single route S_{j_k} of vehicle v_k. The VRP now becomes the problem of choosing at most one column from each block of G so that each row of G has an entry when

$$g_{ij_k} \triangleq \begin{cases} 1 & \text{if customer } x_i \text{ is an element of the single route } S_{j_k} \\ 0 & \text{otherwise} \end{cases}$$

Balinksi and Quandt (as cited by Laporte (1992)) were among the first to propose such a set partitioning formulation for VRPs. But combinatorial problems often result in extremely large arrays of possibilities too complex to be modeled concisely (Rardin, 1998). Column generation adopt a two-part strategy for such problems. It first enumerates a sequence of columns representing viable solutions to parts of the problem, often employing DP. Part two of the strategy solves a set partitioning model to select an optimal collection of these alternatives fulfilling all problem requirements. It results in a flexible and convenient approach employing a multitude of schemes to generate columns which are complex. In this approach it becomes possible to address constraints that are often difficult to model. It suffers, however, from the shortcoming that the number of columns in G can be enormous.

A variant of the VRPTW is solved by Desrochers et al. (1992) who admit that exact solution algorithms have lagged considerably behind the development of heuristics. Their algorithm attempts to use best of breed by solving various subproblems using a branch and bound scheme, DP, and column generation. The drawback remains that the set partitioning problem stops being competitive when a large number of customers are to be serviced on a single route, for example when demands are small in relation to the vehicle capacity. This results in the LP relaxation to become more dense — leading to possible degeneracy.

The MTMCP is closely related to both the TSP and the VRP with the major difference that it is not possible to service all nodes in the graph in the allocated time on a given set of tours. Hence the objective to maximize the earned reward of those nodes visited. In their paper Butt and Ryan (1999) combine column generation and constraint branching to achieve an optimal solution algorithm that solves problems with 100 nodes.

In more recent work Righini and Salani (2004) note that a trade-off remains between the time spent on the column generation, and the quality of the lower bound achieved, indicating that research into effective exact algorithms remain active. Choi and Tcha (2006) use a column generation approach in solving the HVRP with a maximum of 100 nodes in the test problems used. Column generation, however, is not easily adapted to the stochastic variant of a routing problem (Lambert et al., 1993).

3.2 A case for heuristics

Maffioli (1979) indicates that real life combinatorial problems have a number of *unpleasant features*: problems are usually dimensionally large; problems have integrated constraints; and problems can not always be decomposed or generalized to simpler subproblems. It is noteworthy that although researchers attempt to solve real-world problems, complex problems are already solved in industry where decision makers often settle for *good enough* solutions (Russell and Norvig, 2003).

3.2.1 Route construction

Savings-based heuristics

Christofides et al. (1979) indicate that the majority of heuristics are constructive in nature in the sense that at any given stage one or more incomplete routes exist. Incomplete routes are extended to consecutive stages until a final route exists. The construction of routes may be either *sequential* if one route is completed prior to another being started, or *parallel* where more than one incomplete route may exist at a particular stage. After routes are created, a number of local improvements may be initiated to refine a route.

The savings algorithm established by Clarke and Wright (1964) is without doubt the most widely known heuristic in VRPs and has formed the basis of a substantial number of heuristic variations. The Clarke-Wright algorithm remains a computationally efficient algorithm, and deserves attention (Lenstra and Rinnooy Kan, 1981). The algorithm is defined as follows:

Step 1 Calculate the savings for all pairs of customers i and j, denoted by s_{ij}, where both customers are serviced on one route, as opposed to customer i being serviced on a new dedicated route from the depot, using (3.1)

$$s_{ij} = c_{0i} - c_{ij} + c_{j0} \qquad \forall i,j \in \{1,2,\ldots,N\} \tag{3.1}$$

where N is the total number of customers in the network, and c_{ij} denotes the cost of traveling from node i to node j, and where $i = j = 0$ represents the depot.

Step 2 Arrange the savings in descending order of magnitude.

Step 3 Starting from the top, use one of the following approaches:

Sequential approach

1. Find the first feasible link in the list which can be used to extend one of the two ends of the currently constructed route.
2. If the route cannot be expanded, or no route exist, choose the first feasible link in the list to start a new route.
3. Repeat (1) and (2) until no more links can be chosen.

Parallel approach

1. If making a given link results in a feasible route according to the constraints of the VRP, add the given link to the solution. If not, reject the link.
2. Try the next link in the list and repeat (1) until no more links can be chosen.

Step 4 The links form the solution to the VRP.

Christofides et al. (1979) suggest that in the parallel approach a maximum number of routes, M, be introduced to ensure that vehicle feasibility constraints are adhered to. Mole and Jameson (1976) motivate why a sequential approach yields more benefit and adapt the savings procedure to calculate the best insertion position on edge (i,j) of the partially constructed route C for customer u, denoted by $s(i,u^\star,j)$, using the expression in (3.2)

$$s(i,u^\star,j) = \min_{i,j\in C}\{s(i,u,j)\} \qquad \forall u \in \{1,2,\ldots,N\}|u \ni C \tag{3.2}$$

where C is the subset of the N nodes already routed, with

$$s(i,u,j) = 2d_{0u} + (d_{ij} - d_{iu} - d_{uj}) \tag{3.3}$$

The criteria used to determine the best edge to insert a specific customer is referred to as the *insertion criteria*. Once the best edge for insertion has been identified for each customer, the customer with the highest saving will be selected and inserted in its best position. The criteria used to select the best customer is referred to as the *selection criteria*.

Although a number of schemes have been suggested by Christofides et al. (1979) to identify the first customer on a new route, termed the *seed customer*, such as customer with earliest time window deadline; unrouted customer furthest from depot; or customer with largest demand, this thesis will propose a new method for identifying seed customers in Chapter 4.

Nelson et al. (1985) review and test a number of data structures to employ when implementing the Clarke-Wright algorithm. The authors establish methods of choice for VRPs with given characteristics of the network topology.

The savings heuristic has since its inception been adapted in quite a number of research contributions. Golden et al. (1984) refer to the basic savings algorithm as Clarke-Wright (CW), and introduced minor changes through their Combined Savings (CS) algorithm. They proceeded to introduce both the Optimistic Opportunity Savings (OOS) and Realistic Opportunity Savings (ROS). The latter was extended to the ROS-γ that included variety into the algorithm. Solomon (1987) not only applied the savings technique in solving the VRPTW, but also established benchmark problems which have since been used extensively. Paessens (1988), Salhi and Rand (1993) and Tung and Pinnoi (2000) propose various adaptions to the savings heuristic and apply the algorithms to generate feasible routes prior to an improvement stage. In a banking application Lambert et al. (1993) use the savings algorithm on both a deterministic and stochastic variant of the VRPTW. Dullaert et al. (2001) continue the development and adapt the original criteria for sequential insertion, referred to as the Adapted Combined Savings (ACS), Adapted Optimistic Opportunity Savings (AOOS), and the Adapted Realistic Opportunity Savings (AROS).

Liu and Shen (1999b) challenge the prior research by stating that a parallel approach to route construction actually yields superior results, and use the savings algorithm in solving the VRPMVTTW.

Ong et al. (1997) introduce new selection criteria and use the sequential approach on a variant of the Multi Period Vehicle Routing Problem (MPVRP) with time windows, specific vehicle type constraints, multiple depots and stochastic demand constraints. Liu and Shen (1999a) considered the FSMVRPTW and introduced some modifications on the savings expressions with added route shape parameters.

The basic Clarke-Wright algorithm is adapted by Hill et al. (1988), Ahn and Shin (1991), Hill and Benton (1992) and Malandraki and Daskin (1992) to accommodate forward scheduling where time-dependent travel times are modeled. Fleischmann et al. (2004) test three saving algorithms on a time-dependent travel time variant of the VRPTW.

Sweep algorithm

A different approach was introduced by Gillett and Miller (1974). Their proposed algorithm divides the locations into a number of routes. The following notation is introduced to explain the algorithm. Let:

$N \triangleq$	number of locations including the depot (where the depot is always referred to as location 1)
$q_i \triangleq$	the demand at location i, where $i \in \{2, 3, \ldots, N\}$
$(x_i, y_i) \triangleq$	rectangular coordinates of the i^{th} location, where $i \in \{1, 2, \ldots, N\}$
$C \triangleq$	the capacity of each vehicle
$d_{ij} \triangleq$	the distance between locations i and j, where $i, j \in \{1, 2, \ldots, N\}$
$\angle_i \triangleq$	the polar coordinate angle (measured from the depot) of the i^{th} location, where $i \in \{2, 3, \ldots, N\}$
$r_i \triangleq$	the radius from the depot to location i, where $i \in \{2, 3, \ldots, N\}$

The polar coordinate angle is calculated through (3.4).

$$\angle_i = \arctan\left[\frac{y_i - y_1}{x_i - x_1}\right] \tag{3.4}$$

This results in $-\pi < \angle_i < 0$ if $y_i - y_1 < 0$, and $0 \leq \angle_i \leq \pi$ if $y_i - y_1 \geq 0$. The locations are renumbered in ascending order according to the size of their polar coordinate angle such that

$$\angle_i < \angle_{i+1} \qquad\qquad \forall i \in \{2, 3, \ldots, N-1\}$$

The forward sweep portion of the algorithm partitions locations into routes beginning with the location with the smallest angle. Locations are added until the vehicle's capacity is reached, or a preset distance constraint on a route is reached. Subsequent routes are generated in a similar manner until all locations are routed. Each route is then optimised using either exact or heuristic algorithms for the TSP. The minimum distance traveled is then the sum of the distances of each optimised route.

The x-y axis is then rotated counterclockwise so the first location becomes the last, the second becomes the first, the third the second, etc. The minimum distance is calculated again. The rotation of the x-y axis and the calculation of the distance traveled is repeated for all possible axis configurations. The minimum forward sweep distance is the least total distance traveled taken from all axis configurations that was calculated.

The backward sweep portion is similar to the forward portion except that it forms the routes in reverse order, i.e. it start with the last reordered entry based on the polar coordinate angle.

Gillett and Miller (1974) state that the two portions often produce different routes and minimum distances traveled, hence the sweep algorithm's result is the route with the lowest distance traveled of the two portions.

Generalized assignment

Where assignment problems involve the optimal pairing of objects of two distinct types, for example exactly one job order to exactly one machine, or exactly one customer to exactly one sales representative, the *generalized* assignment problem allows for each object i to be assigned to some j, and each j being allowed to receive a number of i (Rardin, 1998). Fisher and Jaikumar (1981) reformulate the VRP in a two-stage approach. First customers are assigned to vehicles, hence the relation to generalized assignment problems. Secondly, for each vehicle the customers assigned to that vehicle is sequenced using the TSP formulation or some other route construction algorithm. The approach is heuristic as the assignment problem's objective function is a linear approximation of the second stage's distance traveled. A number of methodological variants are provided in Nygard et al. (1988). Koskosidis et al. (1992) extend the approach to solve a time window variant of the routing problem.

Giant tours

In the VRP version, a giant tour, including the depot, is first created. A giant tour is a single tour that starts from the depot, passes through *all* customer sites and returns to the depot. A directed cost network is then constructed. Define the tour T_{ab} as a tour beginning with an arc from the depot to customer a, then following the giant tour between customers a and b (which might include other nodes), finishing with an arc from customer b to the depot. There exist a directed edge in the cost network from a to b if and only if the tour T_{ab} is feasible in terms of vehicle capacity and distance restriction. The length of the edge ab in the cost network is the length of T_{ab}. The shortest path problem is subsequently solved using Dijkstra's (1959) algorithm, providing a partitioning of the giant tour.

The procedure is repeated starting from different giant tours and the overall least cost solution is chosen. In their experiments Nagy and Salhi (2005) constructed 5 giant tours; one using the nearest neighbor, another using the least insertion cost rule, and the remaining three tours are generated randomly. A detailed description on how to generate these giant tours and how to construct the associated cost networks can be found in Salhi et al. (1992).

3.2.2 Route improvement

Numerical search is the process of systematically trying different values for the decision variables in an attempt to find a better solution. The process keeps track of the feasible solution with the best objective function value found thus far, referred to as the *incumbent* solution. Rardin (1998) states that most optimization procedures can be thought of as variations of a single theme: *improving search*. Synonyms of the theme include *local improvement*, *hill climbing*, *local search*, and *neighborhood search*.

An improving search heuristic for vehicle routing and scheduling usually starts with a feasible solution created through the route construction heuristics suggested in Section 3.2.1. A characteristic, and unfortunately a drawback of an improving search heuristic is that it advances along its search path of feasible solutions only while the objective function value improves. The search space in which new solutions are investigated is best explained through the analogy of a neighborhood: nearby points of the current solution, each within a small distance from the current solution.

Slight modifications to the current route are referred to as *perturbations*, and are accepted if they yield feasible solutions with an improved objective function value. Although the discussion in this section is by no means exhaustive, it introduces some of the basic mechanisms for creating perturbations. Authors such as Nagy and Salhi (2005) apply combinations of these perturbations sequentially to obtain improved solutions. For purposes of this discussion nodes will be denoted by a, b, c, etc., and routes by bolded characters $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$, etc.

Route reversal

A procedure introduced by Nagy and Salhi (2005) in their Vehicle Routing Problem with Pickups and Deliveries (VRPPD). They observed that changing the direction of a route does not lead to an increase in the route length, and may lead to increased feasibility. In their application the objective is to minimize the infeasibilities when integrating both pickups and deliveries simultaneously, as opposed to sequentially.

2-Opt

A routine introduced by Lin (1965) based on interchanging two edges, say ab and cd, to form two new edges ac and bd.

3-Opt

A modification of the *2-Opt* routine. In this case three arcs are exchanged with three other edges.

Shifting node

Similar to the *3-Opt* routine involving two routes. A single node a is removed from a route $\boldsymbol{x}$ and inserted into another route $\boldsymbol{y}$.

Exchanging nodes

An extension of the *Shifting node* routine. A node a is identified on route $\boldsymbol{x}$, and node b on route $\boldsymbol{y}$. The two nodes a and b are exchanged in their respective positions.

λ-Interchange

When an equal number of nodes, λ, are exchanged between two routes, the perturbation is referred to as λ-Interchange (Tan et al., 2001c; Thompson and Psaraftis, 1993). The *Exchanging nodes* perturbation is therefor a special case where $\lambda = 1$.

Double shift

A more complex extension of the *Shifting node* routine where two nodes, a and b, and three routes, $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{z}$, are considered. Node a is removed from route $\boldsymbol{x}$ and inserted into route $\boldsymbol{y}$, while node b is removed from route $\boldsymbol{y}$ and inserted into route $\boldsymbol{z}$. This is different from performing the *Shifting* routine twice, as after the first *Shift* the resulting route may be infeasible. It should be noted that this routine is computationally more complex as the possible combinations to consider increases substantially.

Splitting a route

According to Mosheiov (as cited by Nagy and Salhi (2005)) a route can be improved if the depot is reinserted into the route, resulting in two routes being created from the original one route considered.

Combining routes

If feasible, two routes $\boldsymbol{x}$ and $\boldsymbol{y}$ are combined, considering both orders $\boldsymbol{xy}$ and $\boldsymbol{yx}$.

3.3 Metaheuristics

The improving search heuristics discussed in the previous section are applied until there are no solutions in the immediate neighborhood hat include a solution that is both feasible and improving. The incumbent solution is then referred to as a *local optimum.* The advantage of heuristics is that good feasible solutions can still be found even though optimality can not be guaranteed; the disadvantage is that uncertainty exists about how close the solutions actually came to the optimal. Herein lies the drawback of heuristics, as the initial solution may negatively influence the optimality of the local optimum found. Refer to the overly simplified illustration in Figure 3.1 and note that if the heuristic starts with a solution at

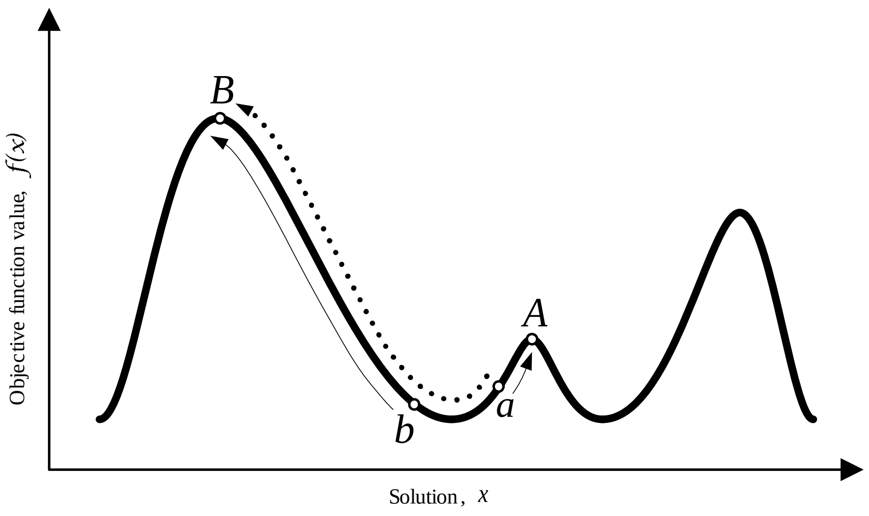

Figure 3.1: Local vs. global optimum

a, it can only improve until it reaches the local optimum at A. If the same heuristic starts with a solution at b it can reach the local optimum at B which also happens to be the *global optimum*: a feasible solution such that no other solution has a superior objective function value.

The interested reader is referred to the TOP Program (2006) research group within the *Foundation for Scientific and Industrial Research at the Norwegian Institute of Technology* (SINTEF). The group has an extensive bibliography of research contributions in the field of vehicle routing, with the majority being on metaheuristics and future research opportunities.

Metaheuristics are master strategies which uses intelligent decision making techniques to guide algorithms to find global optimal solutions by temporarily allowing moves in the neighborhood that result in solutions with inferior objective function values compared to the incumbent solution. Such a deteriorating move is illustrated in Figure 3.1 with a solution starting at a, and deteriorating towards b along the dotted line before improving towards B. A problem arises with accepting temporarily deterioration moves. Consider a and b to be neighbor solutions. Each solution can thus be reached from the other with a single perturbation. A move from a to b may be accepted as a temporarily deteriorating move. However, a move from b to a will always be accepted as it improves the objective function. This may lead to indefinite cycling around a single solution. All metaheuristics follow a similar process, although their specific naming conventions, analogies and detailed routines may vary.

Initialization The process of finding an initial solution. Some metaheuristics, such as the *Genetic Algorithm* performs well with randomly generated initial solutions that need not be feasible, while the *Tabu Search* is highly sensitive to the quality of the initial solution.

Diversification The mechanism that ensures that the underlying heuristics are searching a diversified neighborhood, and thus not getting trapped within local optima.

Intensification A mechanism that ensures the heuristic starts *zooming in* towards a single solution. The most promising neighborhoods are identified and those areas of the solution space are searched more thoroughly.

The diversification and intensification can repeat indefinitely, and hence requires a stopping criteria to terminate the metaheuristic (Van Breedam, 2001). The longer the metaheuristic is run, the higher the probability of converging to the global optimum.

An elementary metaheuristic would entail running the improving search heuristic with multiple initial solutions, each yielding a single local optimum. The best of these local optima is the incumbent solution, denoted by $\hat{x}$. Such a process would be highly reliant on the choice of initial solutions and may yield inferior local optima if initial solutions are not selected and generated carefully.

Four promising metaheuristics are introduced, and interested readers can refer to Gendreau et al. (1998) for a general review of metaheuristics for the VRP.

3.3.1 Tabu Search (TS)

The TS is a memory-based local search metaheuristic introduced by Glover (1986) that searches the neighboring solution space (neighborhood) in search of an improving solution, updating the incumbent solution when improving moves are made. Deteriorating moves are allowed and the metaheuristic deals with cycling by declaring moves to recently visited solutions as *tabu*, hence the name. A thorough and recent review of the TS can be found in Bräysy and Gendreau (2001), where the authors focus on time window variants of the VRP.

A general TS approach is presented in Algorithm 3.1. An initial solution x^0 and a

Algorithm 3.1: Tabu Search

```
   Input: Initial feasible solution $x^0$; Iteration limit $t^{\max}$
 1 $t \leftarrow 0$
 2 $\hat{x} \leftarrow x^t$
 3 Clear Tabu-list, $T = \{\cdot\}$
 4 Generate feasible move set $M^{x^t}$
 5 while either ($\triangle x \in M$ and $\triangle x \ni T$ and $t < t^{\max}$ ) or ($\triangle x$ satisfies aspiration) do
 6     $x^{t+1} \leftarrow x^t + \triangle x$
 7     $T \leftarrow T \cup \{x^{t+1}\}$
 8     if $c\left(x^{t+1}\right) < c\left(\hat{x}\right)$ then
 9         $\hat{x} \leftarrow x^{t+1}$
10     endif
11     $t \leftarrow t + 1$
12     Generate feasible move set $M^{x^t}$
13 endw
```

stopping criteria is required. In this case the stopping criteria is determined to be a preset maximum iteration count $t^{\max}$. The algorithm is initialized by setting the iteration count to zero, setting the initial solution to be the incumbent solution, and clearing the tabu list. The objective function value $c(x)$ is expressed as a function of the solution x. A feasible move set M^{x^t} that represents the neighborhood around the current solution x^t is generated. The neighborhood is established through any of the perturbations discussed in Section 3.2.2. If either no non-tabu move $\triangle x \in M$ leads to a feasible neighbor of the current solution x^t within the preset iteration limit, or some aspiration criteria[1] is met, the metaheuristic

[1]The aspiration criteria may override the tabu list, or the iteration limit criteria.

terminates and the incumbent solution $\hat{x}$ is the approximate optimum. If not, the new neighbor becomes the current solution and is added to the tabu list. The current solution replaces the incumbent if it has a superior objective function value $c\left(x^{t+1}\right)$. The iteration count is incremented, a move set is generated for the new current solution, and the process is repeated.

In a comparison of heuristics and metaheuristics, Van Breedam (2001) identifies TS as a dominant improvement heuristic with the certainty of achieving at least a *local* optimum. Their observation is confirmed by Lee et al. (2006). Ichoua et al. (2003) implement the TS in both a static and dynamic setting, and claim that the model provides substantial improvements over a model based on fixed travel times. Recent developments include drastically reducing the size of the search neighborhood, so called *granular* neighborhoods (Toth and Vigo, 2003). Results obtained when using promising moves, as proposed by granular neighborhoods, yielded good solutions within short computing times.

3.3.2 Simulated Annealing (SA)

As opposed to a local search method, SA is a randomized search method (Brucker, 2004). To understand the concept of simulated annealing in optimization, one has to look at its analogy to the physical annealing system as first introduced by Kirkpatrick et al. (1983). The ground state of a solid, for example steel, is that state in which its atoms or particles are arranged into a minimum energy configuration – the most stable state of the solid. The ground state of a metal can be obtained through the process of physical annealing. The metal is first heated to a high temperature to induce its transformation from a solid to a liquid. This temperature is called the *melting point* of the metal. In its liquid state the metal is unstable, the particles move about freely, exhibiting high energy, since they are not arranged in any set configuration. The temperature is then carefully reduced to allow the particles to gradually settle into the arrangement of minimum energy and the ground state is obtained.

Similarly, SA is aimed at obtaining the minimum value of the objective function of an optimization problem which corresponds to the ground state of the solid (Tan et al., 2001c). Any other state of the solid corresponds to a feasible solution for the optimization problem, and the energy of a state of the solid is equivalent to the objective function value of a solution. A control parameter q, analogous to the temperature of the physical system, is used to control the gradual convergence of the SA algorithm towards the global optimum by regulating the acceptance of moves that deteriorates the objective function value. Similar to a small displacement of the atoms of the solid, the current solution at stage t, x^t, undergoes small perturbations $\triangle x$ as it converges towards the optimum solution.

The general SA metaheuristic provided in Algorithm 3.2 requires an initial solution x^0, and a stopping criteria. Alfa et al. (1991) indicate that the computational time required for finding good solutions are sensitive to the quality of initial solutions. As for the TS algorithm, an iteration limit count $t^{\max}$ is used. The algorithm also requires an initial temperature q^0 and a cooling parameter δ that reduces the temperature of the system after a sufficient number of iterations, denoted by $q^{\max}$.

Initializing the SA algorithm entails setting both the iteration count and the temperature control count to zero, setting the temperature to the initial temperature, and assigning the initial solution as the incumbent $\hat{x}$. The neighborhood is established through any of the perturbations discussed in Section 3.2.2.

If either the iteration count limit $t^{\max}$ is reached, or there are no more feasible moves $\triangle x$ in the neighborhood move set M^{x^t} for the current solution x^t, the algorithm terminates. Otherwise the move is tested for acceptance. If the move is improving the objective function value, it is accepted with a probability of 1. If the move is deteriorating, it will still be accepted with probability

$$P\left[accept\right] = e^{\left(\frac{c(\hat{x})-c(x')}{q}\right)}$$

Returning to the analogy between the physical annealing of a solid and the simulated annealing algorithm, the acceptance criterion for the SA algorithm is deducted from the Metropolis criterion. The Metropolis algorithm, as introduced by Metropolis et al. (as cited by Aarts and Korst (1989)) is a simple algorithm for simulating the physical annealing of a solid. It states that, given a current state i of the solid with energy E_i, a subsequent state j, with energy E_j is generated via a small displacement of the atoms of the solid. If the resulting energy difference, $E_j - E_i$, is less than or equal to zero, j is accepted as the new current state. If, however, the energy difference should be greater than zero, the state j will only be accepted with probability

$$P\left[accept\right] = e^{\left(\frac{E_i-E_j}{k_B T}\right)}$$

where T is the current absolute temperature of the solid and k_B is known as the physical *Boltzmann constant*. Kirkpatrick et al. (1983) noted that since the temperature is merely a control parameter, the Boltzmanns constant can be omitted. The control parameter is

Algorithm 3.2: Simulated Annealing

```
    Input: Initial feasible solution $x^0$; Iteration limit $t^{\max}$
    Input: Initial temperature $q^0 \gg 0$; Temperature limit $q^{\max}$; Cooling factor $0 < \delta \leq 1$
 1  $t \leftarrow 0$
 2  $q^{\text{count}} \leftarrow 0$
 3  $q \leftarrow q^0$
 4  $\hat{x} \leftarrow x^t$
 5  Generate feasible move set $M^{x^t}$
 6  while $\triangle x \in M^{x^t}$ and $t < t^{\max}$ do
 7      $x' \leftarrow x^t + \triangle x$
 8      if $q^{count} = q^{\max}$ then
 9          $q \leftarrow \delta q$
10          $q^{\text{count}} \leftarrow 0$
11      else
12          $q^{\text{count}} \leftarrow q^{\text{count}} + 1$
13      endif
14      if either $c(x') < c(\hat{x})$ or $Probability\left(e^{\frac{c(\hat{x})-c(x')}{q}}\right)$ then
15          $x^{t+1} \leftarrow x'$
16          if $c\left(x^{t+1}\right) < c(\hat{x})$ then
17              $\hat{x} \leftarrow x^{t+1}$
18          endif
19          else
20              $x^{t+1} \leftarrow x^t$
21          endif
22      $t \leftarrow t + 1$
23      Generate feasible move set $M^{x^t}$
24  endw
```

formulated so as to allow virtually all deteriorating moves during the initial stages of the algorithm. As the control parameter is gradually decreased, the probability of accepting deteriorating moves also decreases, and the algorithm converges to the global optimum.

Robusté et al. (1990) indicate in their application of SA that a human can actually outperform the algorithm for large problems in terms of the quality of the solution. Development of the SA have since continued with Van Breedam (1995) reviewing and comparing variants of the SA. Tan et al. (2001c) attribute a number of advantages to the SA metaheuristic:

- Deals with arbitrary systems and cost functions.
- Statistically guarantees an optimal solution (provided sufficient processing time).
- Relatively easy to code, even for complex problems.
- Generally gives a good solution within reasonable processing time.

The latter point has been supported by Van Breedam (2001) stating that the difference in solution quality between TS and SA never exceeded 4% in his evaluation. In their comparative analysis of three metaheuristics, Tan et al. (2001c) conclude that SA is a good compromise between computational effort and quality of solution.

3.3.3 Genetic Algorithm (GA)

GAs were developed and published by John Holland in 1975. GAs are algorithms that search for global optimal solutions by intelligently exploiting random search methods, emulating biological evolution (Rardin, 1998). The relationships between genetic evolution and optimization are:

- *Populations* are represented by groups, each representing a feasible solution.
- In a population, parents mate according to *natural selection.* This is analogous to randomly selected feasible parent solutions.
- *Offspring* are produced by the mating of the selected parents and represent newly created solutions.
- In nature, offspring exhibit some characteristics of each parent since *chromosomes* are exchanged to form new chromosome strings. The algorithm draws on the analogy by creating two new offspring solutions using perturbations such as swapping, on parts of the parent solutions. In GAs the perturbations are often referred to as *crossovers.*

- *Survival of the fittest* is also incorporated as the fitness of a solution can be related to its objective function value. The fittest solutions will typically reproduce to ensure the survival of the fittest solution in the next generation.
- *Mutation* for diversity is represented in the metaheuristic by the random modification of chromosomes, i.e. possible solutions.

Goldberg (1989) reviews GA applications in search strategies an optimization. The general GA metaheuristic provided in Algorithm 3.3 indicates p unique feasible initial solutions

Algorithm 3.3: Genetic Algorithm

```
   Input: Generation limit $t^{\max}$
   Input: Population size $p$; Initial feasible solutions $x_1^0 \dots x_p^0$
   Input: Population subdivisions $p_e$, $p_i$, and $p_c$ such that $p_e + p_i + p_c = p$
1  $t \leftarrow 0$
2  while $t < t^{\max}$ do
3      begin elite
4          Copy $p_e$ best solutions from generation $t$ to generation $t+1$
5      end
6      begin immigrant
7          Include $p_i$ new solutions in generation $t+1$
8      end
9      begin crossover
10         Choose $\frac{p_c}{2}$ non-overlapping pairs of solutions from generation $t$
11         Perform crossover perturbations
12         Include new solutions in generation $t+1$
13     end
14     $t \leftarrow t+1$
15 endw
16 $x^\star \leftarrow \min_{i \in \{1,\dots,p\}} \{x_i^t\}$
17 $\hat{x} \leftarrow$ locally optimized $x^\star$
```

required to constitute generation 0. Filipec et al. (1998) test their GA with various population sizes and conclude that too small a population may terminate the algorithm prematurely as diversification is compromised, while too large populations slows down the convergence rate as more generations are required (increased computational effort) to initiate dominance of quality solutions. Initial solutions are created either randomly or using route construction heuristics as discussed in Section 3.2.1 (Vas, 1999). Skrlec et al. (1997) and Tan et al. (2001c) suggest using only heuristics so as to improve the rate of convergence.

The algorithm only terminates when a sufficient number of generations have existed. Survival of the fittest is ensured as the p_e best solutions of generation t is *cloned* exactly into generation $t+1$. A number, p_i, of new *immigrant* solutions are generated and included in generation $t+1$. The balance of generation $t+1$ is made up by performing various crossover perturbations on a random selection of $\frac{p_c}{2}$ solutions from generation t.

Two distinct approaches are found in literature to solve constrained VRPs with GAs.

Cluster first, route second

This approach was popular in early writings. Thangiah et al. (1991) developed *GIDEON*, a GA program used to solve the VRPTW. At the time it was the best algorithm available for the VRPTW as it produced the best known solutions for 41 of the 56 benchmark problems introduced by Solomon (1987). *GIDEON* has two distinct modules:

Clustering This module assigns customers to specific vehicles in a process called *genetic clustering*. It uses a GA to sector customers into clusters, with each cluster serviced by one vehicle. Figure 3.2 shows the sweeping motion that is is used together with seed angles to create clusters. Each vehicles cluster is routed to minimize route cost, not taking into account vehicle capacities or time windows. The first customer per route, referred to as the *seed customer*, is randomly selected out of the cluster, the rest of the route is formed by determining which customer, when inserted in the route, will produce the lowest route cost, i.e. using a savings heuristic. The best set of clusters obtained by this module is transferred to the next module.

Local route optimization Customers are exchanged between clusters to ensure the feasibility of the solution — taking into account time windows and vehicle capacities. To change a customers cluster, its angle is artificially altered. When a cluster is changed, a cheapest insertion algorithm is used to improve the cluster route.

Nygard and Kadaba (1991), Thangiah and Gubbi (1993), Malmborg (1996), Filipec et al. (1997), Skrlec et al. (1997) and Karanta et al. (1999) were among the contributors using the cluster first, route second approach. Nygard and Kadaba (1991) found that GAs for VRPs tend not to perform well when customers are geographically clustered and a small fleet is

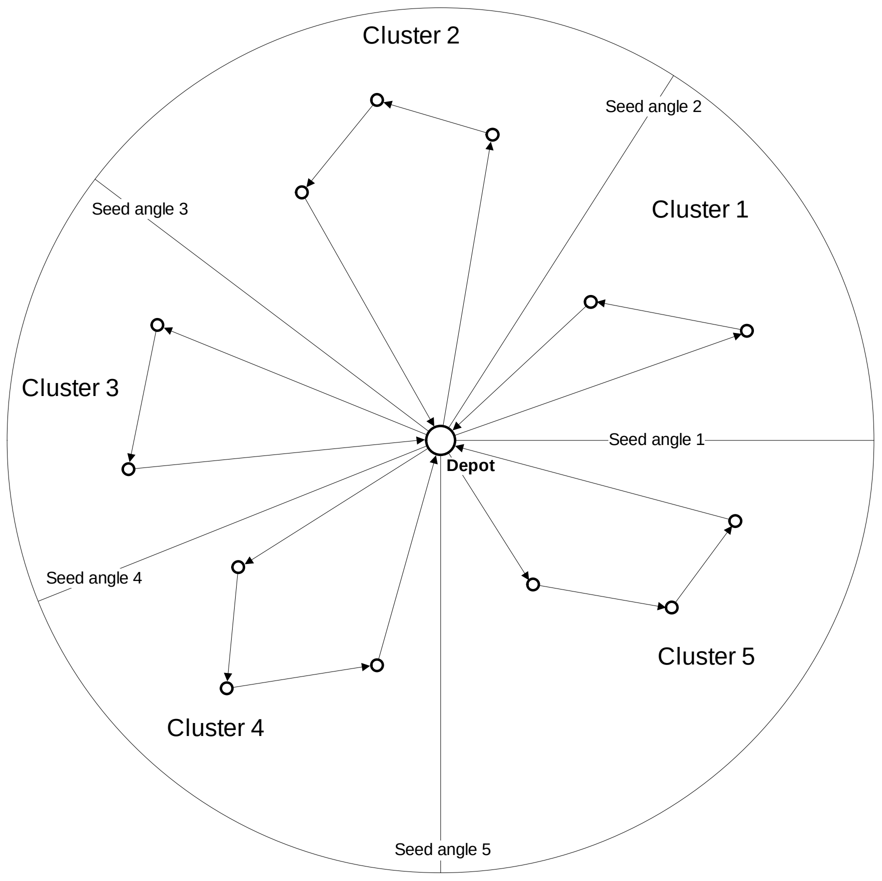

Figure 3.2: Division of customers using seed angles (Thangiah et al., 1991)

used. For all other problem instances the GA performs well. Tan et al. (2001c) claims that the approach *"is only a hybrid heuristic that constitutes some GA element"*.

Route first, cluster second

Recently, path representations are implemented more often for all VRP variations (Filipec et al., 1998; Hwang, 2002; Maeda et al., 1999; Ochi et al., 1998; Prins, 2004; Tan et al., 2001b,c; Zhu, 2003). To indicate separate routes in a chromosome, extra partitioning characters need to be inserted into the chromosome. These extra characters may render the GA useless. GAs use two phases to solve VRP variations, with each chromosome representing a specific path through all the customers. In the first *routing* phase, the *GA* improves the long chromosome string by solving a TSP for all customers. The second *clustering* phase creates a route for each vehicle out of the long route. This is done by another algorithm that adds customers to a vehicle only if time windows are not violated, until the vehicle is full. The following customer in the single route chromosome is then assigned to the next vehicle.

Tan et al. (2001c) were the first to compare three popular metaheuristics, TS, SA and GA for VRP variants. They conclude that GAs were successful in solving the VRPTW, but deduce that there is still no single metaheuristic generic enough to solve all routing problems.

3.3.4 Ant Colony Optimization (ACO)

ACO algorithms are classified as iterative, probabilistic metaheuristics for finding solutions to combinatorial optimization problems. ACO is a general term proposed by Dorigo and Stützle (2002) that includes all ant algorithms. The ant algorithm is an evolutionary approach where several generations of artificial ants search for good solutions. Every ant of a generation builds a solution in a step by step manner, going through several decisions. Ants that found good solution(s) mark their paths through the decision space by placing *pheromone* on the edges of the path. The ants of the next generation are attracted to pheromone and they are more likely to search the solution space near good solutions (Middendorf et al., 2002).

Ant algorithms are inspired by the foraging mechanism employed by real ants attempting to find a shortest path from their nests to food sources. A foraging ant will mark its path by distributing an amount of pheromone on the trail, thus encouraging, but not forcing, other foraging ants to follow the same path (Dorigo et al., 1999). Pheromone is the generic name for any endogenous chemical substance secreted by an organism to incite reaction in other organisms of the same specie. This principle of modifying the environment to induce a change in the ants' behavior via communication is known as *stigmergy*. The effect of stigmergy provides the basis for the ant foraging behavior and artificial ant metaheuristics. Dorigo et al. (1999) discuss the experiments conducted that suggest that the social structure of ant colonies can determine shortest paths between the nest and food sources. A formal proof, however, is absent.

There are a number of direct relationships between real foraging ants and artificial ants used in the ACO metaheuristic.

Colony of cooperating individuals Similar to real ants, artificial ants are composed of a population (or colony) of concurrent and asynchronous entities cooperating to find food timeously. The artificial food are good *solutions* to the optimization problem under consideration. Although the complexity of each artificial ant is such that it can build a feasible solution, high quality solutions are the result of the cooperation among the individuals of the whole colony. This is analogous to a real ant that can by chance find a path between the nest and the food. But only the cooperation of the whole colony can ensure that sufficient food sources are located as close as possible to the nest. Ants cooperate by means of the information they concurrently read and write on the problems states.

Pheromone trail and stigmergy Artificial ants modify some aspects of their environment as real ants do. While real ants deposit a chemical substance, pheromone, on the world state they visit, artificial ants change some numeric information locally stored in the problem state they visit. This information takes into account the ants current history or performance and can be read and written by any ant accessing the state. By analogy, this numeric information is called the artificial pheromone trail, *pheromone trail* for short. In ACO algorithms local pheromone trails are the only communication channels among the ants. This stigmergetic form of communication plays a major role in the utilization of collective knowledge. Its main effect is to change the way the environment (the problem landscape) is locally perceived by the ants as a function of all the past history of the whole ant colony.

Usually, in ACO algorithms an evaporation mechanism is employed, similar to real pheromone evaporation, that modifies pheromone information over time. Pheromone evaporation allows the ant colony slowly to forget its past history so that it can direct its search toward new directions without being over-constrained by past decisions, hence addressing the diversification issue raised for metaheuristics in general.

Shortest path searching and local moves Artificial and real ants share the common task of finding a shortest (minimum cost) path joining an origin (nest) and destination (food). Real ants systematically walk through adjacent terrains' states. Similarly, artificial ants move step-by-step through the neighborhood of solutions of the problem. The exact definitions of state and neighborhood are problem specific.

Stochastic and myopic state transition policy Artificial ants, as does real ants, build solutions applying a probabilistic decision policy to move through adjacent states. As for real ants, the artificial ants' decision policy makes use of local information only and it does not make use of lookahead to predict future states. Therefore, the applied policy is completely local, in space and time. The policy is a function of both the *a priori* information represented by the problem specifications (equivalent to the terrains structure for real ants), and of the local modifications in the environment (pheromone trails) induced by past ants.

Artificial ants also have some characteristics that do not have counterparts in real ants. Artificial ants live in a discrete world and their moves consist of transitions from discrete states to discrete states. Artificial ants have an internal state. This private state contains the memory of the ants' past actions. Artificial ants deposit an amount of pheromone that is a function of the quality of the solution found. Timing in pheromone laying is problem dependent and often does not reflect real ants behavior. For example, in many cases artificial ants update pheromone trails only after having generated a solution. To improve overall system efficiency, ACO algorithms can be enriched with extra capabilities such as the ability to forecast, local optimization, and backtracking that cannot be found in real ants.

An ant is a simple computational agent, which iteratively constructs a solution for the instance to solve. Partial problem solutions are seen as *states*. At the core of the ACO algorithm lies a loop, where at each iteration, each ant moves (performs a step) from a state i to another one j, corresponding to a more complete partial solution.

Algorithm 3.4 is based on Maniezzo et al. (2004) and requires an *a priori* desirability

Algorithm 3.4: Ant Colony Optimization

```
Input: Attractiveness $\eta_{ij}$; Trail level $\tau_{ij}$
Input: Number of ants $k$
while $t < t^{\max}$ do
    for each ant $k$ do
        repeat
            choose in probability the state $j$ to move to
            append ant $k$'s set $tabu_k$
        until ant $k$'s solution is complete
    endfor
    for each ant move $(i, j)$ do
        compute $\triangle\tau_{ij}$
        update trail matrix
    endfor
    $t \leftarrow t + 1$
endw
```

of the route referred to as the attractiveness, η_{ij}, for each origin-destination pair (i, j). The attractiveness is often also referred to as the *heuristic information* (Meuleau and Dorigo, 2002). The trail level τ_{ij} of the move from i to j is required and indicates how beneficial it has been in the past to make that particular move. The trail level therefore represents an *a posteriori* indication of the desirability of the move. As in previous metaheuristics discussed, an iteration limit $t^{\max}$ terminates the ACO.

For each ant k a solution is incrementally built using both the attractiveness and the trail level, weighted with preset parameters. Each ant's memory of tabu moves are updated accordingly to ensure only feasible solutions are created. Once all ants have their solutions, the pheromone trail matrix is updated by determining how many ants (solutions) traversed specific edges (i, j). The iteration number is incremented, and the process repeated until the maximum number of iterations have been reached.

Although Bullnheimer et al. (1999) could not improve on the best solutions found for sets of benchmark problems, the competitiveness of ACO is applaudable, given the immaturity of the approach to VRP variants compared to established, and well-researched metaheuristics. Detailed algorithmic approaches are provided by Dorigo and Gambardella (1997a,b), and Meuleau and Dorigo (2002) for the TSP and Gambardella et al. (1999) for the VRPTW which should stimulate and accelerate research in the respective fields and its variants. A robust algorithm presented by Reimann et al. (2004) is able to solve a number of VRP variants.

3.4 Conclusion

In the first review article of ACO theory, Dorigo and Blum (2005) comprehensively state that research contributions using metaheuristics as new as the ACO focus on proof-of-concept. This, however, is still true for the majority of theoretical papers on heuristics and metaheuristics. Solution quality and computational burden of various algorithm contributions are compared using benchmark problems (Van Breedam, 2001). The state-of-the-art for generic variants of the VRP are often implemented in commercial software applications. In such applications the parameter values for the specific metaheuristic are usually fixed, and are based on experiments with the benchmark data.

The majority of literature reviewed in this chapter either suggest parameter values that perform well in the majority of cases, or confirm that parameter settings are inherently problem specific. This review concludes with the observation that an intelligent routing system is required that will be able to observe the problem environment in which it is implemented, and dynamically adjust parameter settings in order to improve future solutions.