CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Review of Bio-Inspired Algorithms for Vehicle Routing

Jean-Yves Potvin

July 2008

CIRRELT-2008-30

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince, local 2642
Québec (Québec)
Canada G1K 7P4
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

A Review of Bio-Inspired Algorithms for Vehicle Routing

Jean-Yves Potvin[1,*]

[1.] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-ville, Montréal, Canada H3C 3J7

Abstract. This chapter reviews biologically inspired algorithms for solving a class of difficult combinatorial optimization problems known as vehicle routing problems, where least cost collection or delivery routes are designed to serve a set of customers in a transportation network. From a methodological standpoint, the review includes evolutionary algorithms, ant colony optimization, particle swarm optimization, neural networks, artificial immune systems and hybrids. From an application standpoint, the most popular vehicle routing variants are considered, starting with the classical vehicle routing problem with capacity constraints.

Keywords. Natural computing, biologically inspired algorithms, evolutionary algorithms, ant colony optimization, particle swarm optimization, neural networks, artificial immune systems, vehicle routing.

Acknowledgements. Financial support for this work was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC). This support is gratefully acknowledged.

---

* Corresponding author: Jean-Yves.Potvin@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec,
Bibliothèque et Archives Canada, 2008

1 Introduction

This paper is interested in biologically inspired computing, a branch of natural computing [14] where algorithms inspired from nature are developed to solve highly complex problems, in particular problems that cannot be addressed in a satisfactory way with traditional approaches. Under this paradigm, algorithmic models of processes observed in nature are developed and implemented on computers to explore solution spaces.

In the scientific literature, there is a growing interest in the exchange of ideas between natural systems and computational systems. Bio-inspired algorithms are representative of this trend and have been applied to a large variety of problems, including optimization problems. Among them, vehicle routing problems have generated a lot of attention, because they are inherently complex and hold a central place in real-world distribution activities. They generalize the well-known traveling salesman problem (TSP), a canonical combinatorial optimization problem that has been widely studied in the literature [45]. Given a graph with a number of vertices and a cost associated with each arc, the TSP looks for a least cost tour that visits each vertex exactly once (where the cost of a tour is the sum of its arc costs). In vehicle routing problems, side constraints are introduced to limit the number of vertices that can be visited with a single tour, thus leading to solutions that cover all vertices with different routes that start from and end at a particular vertex, called the depot. These problems are used to model various real-world applications. In the manufacturing sector, for example, they model transportation activities within the supply chain, that is, the movement of goods from suppliers to factories and then to customers. In the service sector, they include activities such as mail delivery, garbage collection, public transportation, etc.

The chapter is organized as follows. Section 2 introduces the basic Vehicle Routing Problem (VRP) and its main variants. This is followed in Section 3 by a brief review of some classical problem-solving methodologies. Then, Sections 4 to 8 introduce the main bio-inspired algorithms, namely Evolutionary Algorithms (EA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Neural Networks (NN) and Artificial Immune Systems (AIS), in this order, and describe their application to vehicle routing problems. Section 9 is then devoted to hybrids that combine ideas from different bio-inspired algorithms. Concluding remarks follow in Section 10.

2 Vehicle Routing

The aim of vehicle routing is to determine optimal collection or delivery routes for a fleet of vehicles in a transportation network [43, 44, 85]. These NP-hard problems generalize the classical Traveling Salesman Problem (TSP) by requiring both an assignment of vertices to vehicles and a sequencing of these vertices within each vehicle route to obtain a solution. A large number of

variants are reported in the literature, depending on the objective function to be optimized and the types of constraints to be satisfied. These variants are reviewed in the following.

2.1 Vehicle Routing Problem (VRP)

The VRP can be formally stated as follows. Let $G = (V, A)$ be a graph where $V = \{1, 2, ..., n\}$ is the vertex set and A is the arc set. Vertex 1 is the depot for a fleet of identical vehicles of capacity Q that collects a demand q_i at each vertex (customer) $i \in V - \{1\}$. A non negative cost c_{ij} is also associated with every arc $(i, j) \in A$, $i \neq j$. This cost is often interpreted as a distance or a travel time, depending on the context. Unless otherwise stated, it is assumed in the following that the problem is symmetrical, that is, $c_{ij} = c_{ji}$ for every arc (i, j). The problem is then to determine a set of least cost vehicle routes such that:

- each vertex, apart from the depot, is visited exactly once by exactly one vehicle to serve its demand;
- all vehicle routes start and end at the depot;
- the total demand on each route does not exceed the vehicle capacity;

Sometimes, a fixed cost is associated with each vehicle. The objective is then aimed at minimizing a weighted sum of fixed costs and travel costs. A maximum distance or time constraint can also be considered, in addition to the capacity constraint, for each vehicle route.

2.2 Vehicle Routing Problem with Time Windows (VRPTW)

In this extension of the VRP, a time interval or time window $[a_i, b_i]$ constrains the beginning of service at each vertex $i \in V - \{1\}$. There is also a window $[a_1, b_1]$ at the depot to constrain the start time and end time of each vehicle route. The upper bound b_i at vertex i can be either a hard of a soft constraint. In the latter case, the vehicle is allowed to arrive late, but a penalty is incurred in the objective. A waiting time is introduced in the route schedule when the vehicle arrives before the lower bound a_i. In most papers, a hierarchical objective is considered. The first objective is to minimize the number of vehicles and, for the same number of vehicles, the distance, travel time or scheduling time (travel time + waiting time) is minimized. The VRP with time Deadlines (VRPD) is a special case of the VRPTW where only a time upper bound is associated with each vertex.

2.3 Vehicle Routing Problem with Backhauls (VRPB)

Here, the set of vertices is partitioned into two subsets: linehaul vertices for which the demand is delivered from the depot and backhaul vertices for which

the demand is picked up and brought back to the depot. It has been quickly recognized that substantial cost savings can be achieved by allowing empty vehicles to pick up inbound products when they return to the depot. In the grocery industry, for example, linehauls could be supermarkets where goods are delivered while the backhauls could be grocery suppliers. In the classical VRPB, there is a strict precedence relationship between the linehaul and backhaul vertices, that is, all linehauls must be visited before the backhauls. Without this constraint, goods could be picked up while other goods would not have yet been delivered, thus potentially leading to a rearrangement of the goods inside the vehicle. In other variants, this constraint is relaxed to allow a vehicle to pick up goods if the capacity utilization is sufficiently low (i.e., if only a few linehaul vertices have not been visited yet). Finally, in the mixed VRPB, linehaul and backhaul vertices can be freely mixed as long as the capacity constraint is satisfied. When time windows are associated with the vertices, the VRPBTW is obtained.

2.4 Time-Dependent Vehicle Routing Problem (TDVRP)

In the TDVRP, the travel times are not fixed but rather depend both on the distance between two vertices and the time of the day (e.g., it takes longer to get from one location to another during rush hours). The TDVRP is thus aimed at more closely modeling situations observed in the real world.

2.5 Pick-up and Delivery Problem (PDP)

In this problem, each customer request corresponds to a pair of vertices. That is, for a given pair of vertices $i+$ and $i-$, the demand must be picked up at vertex $i+$ and delivered at vertex $i-$. Clearly, both vertices must be in the same vehicle route (pairing constraint) and vertex $i+$ must be visited before $i-$ (precedence constraint). When time windows are associated with the vertices, the PDPTW is obtained. When people are transported, as in transportation-on-demand services, trip duration constraints must also be taken into account. The latter problem is known as the dial-a-ride problem (DARP).

2.6 Other Vehicle Routing Problems

Without being exhaustive, we also want to mention the following problems:

- *multiple-depot* problems where each vehicle can start (end) its route from (to) a different depot;
- *mixed fleet and size* problems where the fleet size must be determined based on different types of vehicles with different characteristics (e.g., capacity);
- *periodic vehicle routing* problems where routes are determined over an horizon that spans a number of periods and where each vertex must be visited at a given frequency within this horizon.

- *location-routing* problems where strategic decisions about the location of different facilities (e.g., depots, warehouses) are taken concurrently with the determination of vehicle routes;
- *inventory routing* where each vertex has an inventory and delivery routes are determined to replenish the inventory so as to avoid any inventory shortage.
- *stochastic vehicle routing* problems where some data are not known with certainty. For example, the demand might not be known with certainty at a given vertex, but the probability distribution is known and can be used to minimize the expected solution value.
- *dynamic vehicle routing* problems where new information are obtained as the vehicle routes are executed. For example, new service requests occur and must be integrated into the current routes or the travel times suddenly change due to unforeseen events.

There are also a number of vehicle routing problems where arcs must be visited instead of vertices, but these problems are not addressed here (see [20, 21, 22]). Arc routing problems are useful to model applications where the customers are located along the streets of the transportation network, as in postman delivery routes.

3 Classical Problem-Solving Methodologies

In this section, we briefly review some classical methodologies for solving vehicle routing problems. These methodologies can be divided into two main classes: exact methods and heuristics. Exact methods are divided into (1) tree search methods, like branch-and-bound, (2) dynamic programming and (3) integer programming-based methods. Although these methods produce optimal solutions, they can be computationally expensive, even for relatively small instances. We refer the reader to [43] for further details on this topic.

In the case of heuristics, we can distinguish between construction heuristics, improvement heuristics and metaheuristics, as it is explained below.

3.1 Construction Heuristics

In this category, we have pure construction heuristics and two-phase construction heuristics. Pure construction heuristics are:

Insertion heuristics

These heuristics are widely used to quickly construct a solution. At each iteration, a vertex is selected among all unvisited vertices and a feasible, least cost, insertion place between two consecutive vertices is looked for. This insertion process is repeated until all vertices are visited. This is a generic approach that

can easily account for different types of side constraints. Figure 1 illustrates the insertion of vertex k between two vertices i and j with the corresponding insertion cost or detour $c_{ik} + c_{kj} - c_{ij}$.

In sequential variants, routes are constructed one by one. That is, the vertices are inserted in the current route until it is not possible to add any new vertex without violating one or more constraints. At this point, a new route is constructed. In the parallel variants, all routes are constructed at once. Thus, a vertex can be inserted in any particular route, depending on the location of the best feasible insertion place.

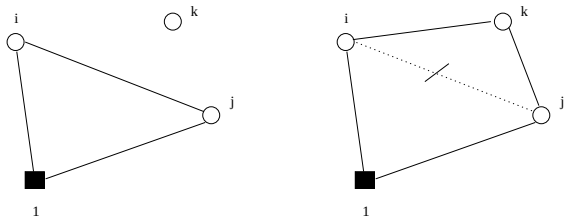

Fig. 1. Inserting a vertex

Savings heuristic

The savings heuristic is a well-known problem-solving approach for the VRP [11]. Starting from a configuration where each vertex is visited by an individual route connected to the depot, two routes are selected at each iteration and merged together, as illustrated in Figure 2. The merging process is guided by the savings measure. That is, for a given pair of vertices i and j, the initial cost of the two individual routes is $2c_{1i} + 2c_{1j}$. By merging these two routes into a single route, the cost becomes $c_{1i} + c_{1j} + c_{ij}$ for a savings $s_{ij} = 2c_{1i} + 2c_{1j} - (c_{1i} + c_{1j} + c_{ij}) = c_{1i} + c_{1j} - c_{ij}$. These savings are thus calculated for every pair of vertices and sorted in non increasing order. Pairs of routes are then merged together, by using the sorted list of savings to guide the merging process (starting with the largest one) until it is not possible to merge two routes without violating the side constraints.

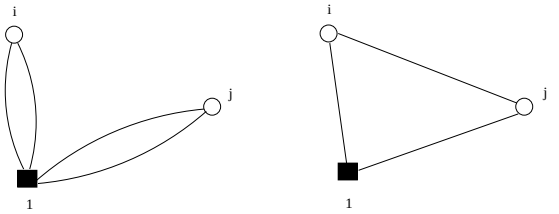

Fig. 2. Merging two routes

Two-phase construction heuristics are:

Cluster-first, route-second

Here, vertices are first grouped into distinct subsets based on some proximity measure. Then, within each group, the vertices are ordered to form a route using, for example, an insertion heuristic.

Route-first, cluster-second

This an alternate approach to cluster-first, route-second, where a giant route that visits all vertices is first constructed. Then, this giant route is partitioned into smaller feasible routes.

3.2 Improvement Heuristics

Once an initial solution has been constructed, it can be further improved with a local search heuristic. In this case, a class of modifications to the current solution is defined to generate a neighborhood of solutions. The best solution in this neighborhood, if better than the current solution, then becomes the new current solution. The procedure is repeated until there is no better solution in the neighborhood of the current solution. Some basic modifications, based either on nodes or arcs in the solution, are the following:

- *vertex move*: a vertex is removed from the solution and inserted at another place;
- *vertex swap*: two vertices exchange their position in the solution;
- *arc exchange*: r arcs are removed from the solution and are replaced by r new arcs to produce a new valid solution. The most widely used exchanges of this type involve $r = 2, 3$ arcs and are called 2-opt and 3-opt [48].

Numerous extensions and adaptations of these basic movements for different types of vehicle routing problems are reported in the literature. For example, it is possible to move or swap sequences of consecutive vertices (rather than a single vertex) or to consider arc exchanges that maintain the current orientation of the routes for problems with time windows. Also, these modifications can be applied to each route individually, to modify the sequence of vertices in the route, or they can be applied to several routes at once to modify the vertex assignment.

3.3 Metaheuristics

The emergence of metaheuristics for solving difficult combinatorial optimization problems, including vehicle routing problems, is one of the most notable achievement of the last two decades in operations research. Metaheuristics offer global search strategies for exploring the solution space that are then

adapted to a particular class of problems. Some metaheuristics are high-level abstraction of optimization processes observed in nature and will be described in the next sections. Among the other types of metaheuristics that have been successful on different types of vehicle routing problems, tabu search [32] is certainly the most notable one. Some applications of simulated annealing [41] and variable neighborhood search [54] are also reported in the literature. In all cases, the definition of an appropriate neighborhood structure is a crucial point. Some researchers favor streamlined implementations with a few parameters and simple neighborhood structures, like the unified tabu search [12]. Others propose complex modifications to the current solution, thus leading to large or very large neighborhood searches [1]. The reader is referred to [27] for further details on this issue.

In the following sections, the application of the main bio-inspired algorithms for solving vehicle routing problems are reviewed, starting with EAs.

4 Evolutionary Algorithms

Evolutionary algorithms are stochastic search methods that operate on a population of solutions by simulating, at a high level of abstraction, the evolution of species observed in nature. Genetic algorithms (GA) [34] were the first evolutionary algorithms to be applied to combinatorial optimization problems. In a GA, a population of solutions evolves over a number of generations through the application of operators, like selection, crossover and mutation, that mimic the corresponding genetic processes observed in nature. Typically, the solutions are encoded using strings of fixed length (in classical GAs, bitstrings are used). More formally, a GA can be sketched as follows:

1. Create an initial population of P solutions.
2. Evaluate each solution.
3. Repeat for a fixed number of generations:
   - 3.1 Repeat until P offspring solutions are created:
     - 3.1.1 Select two parent solutions in the population (with replacement) using a randomized selection procedure based on the solution values.
     - 3.1.2 Apply crossover to the two parent solutions to create two offspring solutions.
     - 3.1.3 Apply mutation (with a small probability) to each offspring.
     - 3.1.4 Include the two offspring in the new population.
   - 3.2 Evaluate each offspring in the new population.
   - 3.3 Replace the old population by the new one.
4. Return the best solution found.

First, an initial population of solutions is created, either randomly or through heuristic means. Then, the value of each solution, called fitness in

GA terminology, is computed. The next population is then created from parent solutions chosen in the current population. A typical randomized selection procedure for the GA is the proportional or roulette wheel selection, where the selection probability of each solution is proportional to its value. When two parents are selected, they are modified, with a certain probability, by the crossover operator. Here, two offspring solutions are created from two parents through an exchange of solution components. For example, in the case of a bitstring encoding, the classical one-point crossover selects a cut point on the two parent strings and exchanges their end parts. Mutation is a secondary operator that processes each offspring position by position and flips the bit value at each position with a small probability. Together, selection and crossover search the problem space by combining solution components associated with good solutions to create new solutions with hopefully better values. Mutation acts as a perturbation operator to prevent premature convergence to bad suboptimal solutions.

Although some theoretical results that characterize the behavior of a GA have been reported for bitstring encodings, not all problems lend themselves easily to this representation. This is the case, in particular, for vehicle routing problems where an integer representation is more appropriate. In this case, an integer stands for a vertex, while a string of integers represents a vehicle route. For example, the string 1-5-2-3-1-4-6-1 would mean that, starting from the depot, one vehicle visits vertices 5, 2 and 3, in this order, before returning to the depot; a second vehicle visits vertex 4 first and then vertex 6 before returning to the depot. Note that the depot index 1 acts as a separator between the two routes. With this representation, called the path representation, a straightforward application of a classical GA does not work. As illustrated in Figure 3, if we assume a single route on six vertices numbered from 1 to 6, where vertex 1 is the depot, the one-point crossover produces two invalid offspring 1-2-3-4-5-6 and 1-6-5-4-3-2 when the cut point is chosen between the 4th and 5th position (note that the return to vertex 1 at the end to close the tour is not shown and is implicit). On both offspring, vertices are duplicated while others are missing.

parent 1:	**1 2 3 4** |	**5 6**
parent 2:	1 6 5 4 |	3 2
offspring 1:	**1 2 3 4** |	3 2
offspring 2:	1 6 5 4 |	**5 6**

Fig. 3. One-point crossover

Numerous studies have thus been conducted to address vehicle routing problems with GAs based on innovative representation schemes and specialized operators. For example, order-based crossover operators, like the Order crossover OX [59], have been designed to allow valid offspring routes to be generated. In the case of OX, two cut points are randomly chosen and the substring between the two cut points on the first parent is copied to the offspring. Then, the remaining positions are filled by following the ordering on the second parent, starting at the position just after the second cross point. When the end of the string is reached, the procedure resumes at position 1. Figure 4 shows an example on a tour with six vertices. The substring made of vertices 3 and 4 in parent 1 is first copied to the offspring. Then, the vertices are processed in the order 3, 2, 1, 6, 5, 4 on the second parent. Vertex 3 is discarded because it is already included in the offspring. Vertex 2 is then inserted at position 5, followed by vertex 1 at position 6, vertex 6 at position 1 and vertex 5 in position 2. Finally, vertex 4 is discarded. This operator thus produces a valid ordering. Many order-based crossover operators like this one are reported in the TSP literature to produce valid offspring orderings [62]. The experience gained with the TSP have also led researchers to use powerful local search-based mutation operators that fully optimize each offspring solution produced through crossover.

parent 1:	**1**	**2**	|	**3**	**4**	| **5 6**
parent 2:	1	6	|	5	4	| 3 2
offspring:	6	5	|	**3**	**4**	| 2 1

Fig. 4. OX crossover

From the previous discussion, it is clear that a number of GAs for solving vehicle routing problems are direct extensions of GAs developed for the TSP. A good example is the edge assembly crossover EAX, an operator that proved very powerful for the TSP and was later adapted to the VRP [57]. In the case of the VRP, this operator can be described as follows:

1. Construct a graph by combining the edges of the two parent solutions.
2. Partition the set of edges in this graph with cycles created by alternately selecting an edge from the first and second parents.
3. Select a subset of cycles.
4. Generate an intermediate solution as follow. Take one parent and remove all edges that are in the selected subset of cycles. Then, the edges in the subset of cycles that come from the other parent are added. At this point, the intermediate solution is a collection of routes connected to the depot plus subtours that are not connected to the depot.

5. Create a complete solution. A greedy heuristic is applied where, at each iteration, a subtour is merged at least cost to a route or to another subtour. The procedure is repeated until a set of routes is obtained that visits all vertices.
6. Eliminate capacity violations. Using a penalty function for route overcapacity, restricted 2-opt arc exchanges [48] and vertex swaps are applied until a feasible solution is obtained. These modifications are said to be restricted because an infeasible route must be involved.

The *EAX* crossover is illustrated in Figure 5. From two parents, a combined graph is obtained. Then, based on the particular set of cycles shown in this example, an offspring is produced. The latter is obtained from parent 1, by removing the edges of parent 1 and by adding the edges of parent 2 that are in the set of cycles. Clearly, the offspring is not a valid solution since it contains two subtours that are not connected to the depot. These subtours thus need to be merged with the routes that are connected to the depot to obtain a valid solution.

A GA with *EAX* reached the best solution on a subset of VRP instances taken from Christofides, Mingozzi and Toth [10] and it found ten new best solutions on the data set of Golden et al. [33]. This work has also been extended to the VRPTW in [58]. In this case, an additional term for time window violations is included in the penalty function when feasibility is restored. Another good example of a crossover operator that has been extended from the TSP to the VRPTW is the natural crossover (for a description of this operator, see [38]).

In the following subsections, we present GA methodologies that have been specifically developed for vehicle routing problems. They are illustrative of different ways of handling side constraints.

4.1 Cluster-First, Route-Second

The methods presented in this section follow the cluster-first, route-second problem-solving approach, where clusters of vertices that naturally fit in the same vehicle route (due, for example, to spatial proximity) are first identified. Then, the vertices are ordered within each cluster to form routes. Here, the GA is used in the clustering phase, while the routing is done through insertion and improvement heuristics.

GIDEON

GIDEON [82] was applied to the VRPTW by first clustering the vertices in a way similar to the sweep heuristic of Gillett and Miller [31]. In the sweep heuristic, a seed point is chosen and, using the depot as a pivot, the ray from the depot to the seed is swept clockwise or counter-clockwise. Vertices are added to the current cluster as they are swept until the capacity constraint forbids the addition of the next vertex. This vertex then becomes the seed

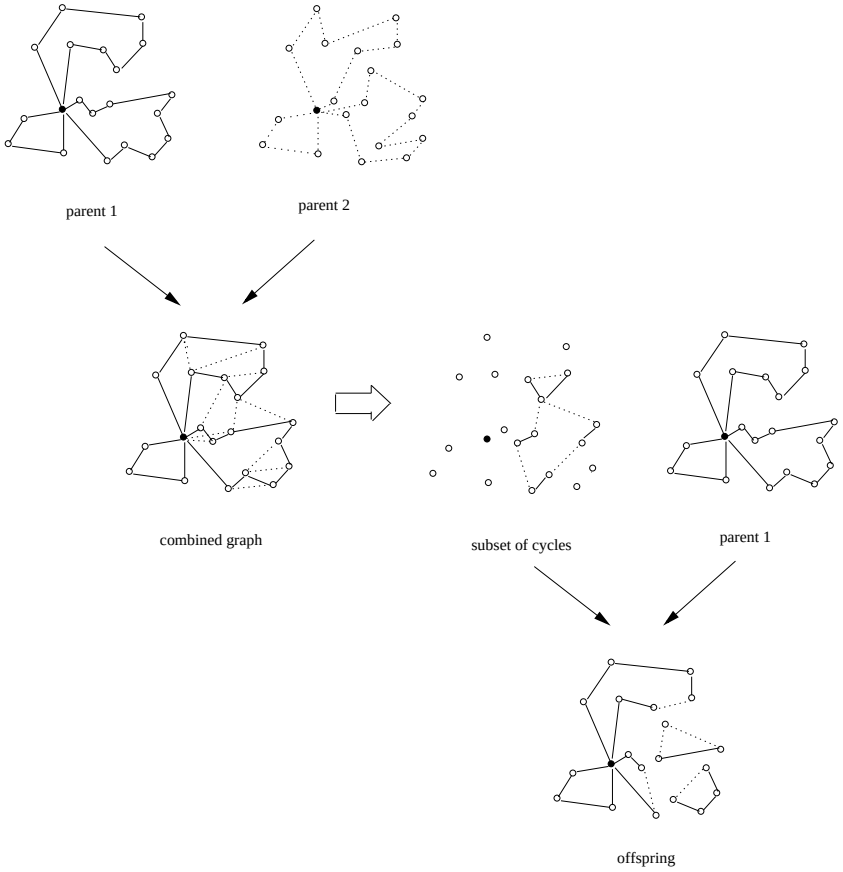

Fig. 5. *EAX* crossover

point for the next cluster. This is repeated until all vertices are assigned to a particular cluster. Routes are then constructed within each cluster by sequencing the vertices.

In the case of GIDEON, we assume a set of n vertices with planar coordinates (x_i, y_i) and polar angle p_i. The method divides the vertices into m clusters by identifying a set of seed angles, s_0, ..., s_m and by drawing a ray between the depot and each seed angle. The initial seed angle s_0 is assumed to be 0^o. The first cluster thus lies between seed angles s_0 and s_1, the second one between seed angles s_1 and s_2, and so on. GIDEON assigns vertex i to cluster k if $s_{k-1} < p_i \leq s_k$, $k = 1,..., m$, as illustrated in Figure 6. Each angle s_k is obtained from the previous one by adding a fixed angle F and an offset e_k. That is, $s_k = s_{k-1} + F + e_k$, $k = 1,..., m$. The fixed angle is the minimum angle value and insures that each cluster is represented. The offset is the extra angle that allows the cluster to encompass a larger or smaller area. If the

addition of the fixed angle F and the offset e_k to s_{k-1} exceeds 360^o, then s_k is set to 360^o, thereby allowing the method to consider less than m vehicles in a solution. Within this clustering scheme, a set of m integer offsets is encoded in base-2 notation on a bitstring, and the GA searches for the offsets that result in the best solution.

Once the clusters are identified, the ordering of the vertices within each cluster is done with a least-cost insertion heuristic, where the cost accounts for the total distance and penalties for time window and capacity violations. Through these penalties, infeasible solutions are less attractive and less likely to be selected as parents for the next generation. At the end, a local search heuristic is applied to the best solution produced by GIDEON. The neighborhood structure is based on λ-interchanges [60], with $\lambda = 2$. In this case, one or two vertices are moved to another route or exchanged with one or two vertices from another route. This final phase is particularly important to reduce or eliminate capacity or time window violations. GIDEON led to the development of GenClust, which is described in the following.

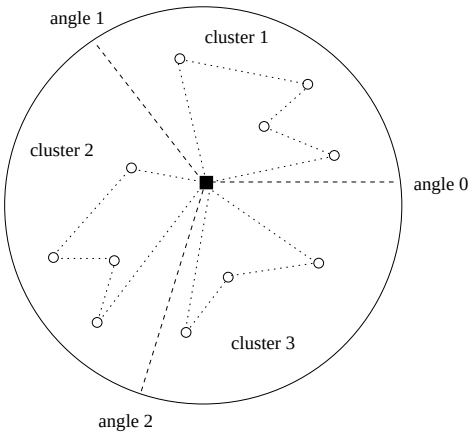

Fig. 6. clustering phase of GIDEON

GenClust

Quite often, it is desirable to have vehicle routes with particular geometric shapes, like petal or circular shapes. GenClust [81] explores this idea, while following the general principles of GIDEON. Here, a geometric shape is described via a set of attributes. For example, two attributes may be used to describe a circle, namely, its center (x, y) and its radius r. These numeric attributes are then encoded on a bitstring in base-2 notation. More precisely, each string encodes m different circles, one for each cluster, so that all vertices within a given circle are assigned to the same vehicle route (when a vertex

is not inside any circle, it is simply associated with the closest one). The GA thus searches for the set of circles that lead to the best solution. GenClust was later applied to a multi-depot extension of the VRP [83]. In this case, for each set of circles produced by the GA, a solution to the multi-depot problem is obtained by first associating each circle or cluster with the closest depot.

4.2 Route-First, Cluster-Second

This approach uses a path representation to encode a (likely infeasible) unique giant tour that visits all vertices. With this representation, classical order-based crossover and mutation operators previously developed for the TSP can be used, given that a single tour is involved. A final solution is obtained by partitioning the giant tour into individual feasible routes. In [65], an exact polynomial-time algorithm is proposed to perform the partition in the case of the VRP. This algorithm works on an auxiliary directed acyclic graph with vertex set {1, 2,..., n}, where vertex 1 is the depot. An arc (i, j) is added to the graph when a route from vertex $i + 1$ to vertex j, based on the ordering in the giant tour, is feasible. The length of the arc is the total length of the corresponding route. A solution is then obtained by solving a shortest-path problem from node 1 to node n in this graph. This can be done in polynomial time due to the acyclic nature of the graph.

4.3 Ruin and Recreate

Here, the GAs are inspired from the ruin and recreate paradigm, where a fraction of a solution is destroyed and then reconstructed in some alternative way [73]. For example, we can choose to remove q vertices from a solution to ruin it. Since each selection of q vertices is likely to lead to a different solution after reconstruction, the ruin and recreate principle can also be used to define a neighborhood structure for a local search heuristic. In this example, the size of the neighborhood grows quickly with q, thus leading to a large neighborhood search.

A good example of this approach is found in the work of Berger and his colleagues for the VRPTW. In [3], for example, an insertion-based crossover operator is proposed. First, a route r is probabilistically selected from the first parent solution, with a bias toward routes with large waiting times. A number of vertices are then removed from route r (ruin), based on criteria indicating that a more suitable relocation of these vertices into alternate routes is likely, due to a large distance to the immediate successor or a large waiting time. Vertices in a subset of routes from the second parent that are close to route r are then inserted, if possible, in the first parent solution. This procedure is repeated for a number of routes in the first parent solution. At the end, the offspring solution is completed with the remaining routes from the first parent. If there are unvisited vertices, they are handled by constructing additional routes with a nearest neighbor heuristic. A second child is obtained

by interchanging the role of the two parents. Three mutation operators are also used to either reduce the number of routes or to reorder the vertices within the routes.

In a follow-up work [2], the authors relax the time constraints and evolve two populations with different objectives. Population 1, which contains at least one feasible solution, minimizes the total distance, while population 2 minimizes the time constraint violations, both subject to a fixed number of routes (i.e., m and $m-1$ routes, respectively). When a feasible solution with $m-1$ routes emerges in population 2, population 1 is replaced by population 2 and a special mutation operator is applied to population 2 to obtain a new population made of solutions with one fewer route. The number of routes is reduced in this way until no feasible solution emerges in population 2. A suite of five mutation operators is also proposed, including a large neighborhood search-based mutation that follows the guidelines in [74]. Here, related vertices (due to proximity or a common route assignment) are removed from the solution and reinserted with a variant of Solomon's I1 insertion heuristic [76].

A Sequence-Based crossover (SBX) is reported in [63] that merges the first part of a route in the first parent solution to the end part of another route in the second parent solution. The new route then replaces the old one in the first parent solution. In the process, vertices are likely to be duplicated or removed. Hence, a repair operator is applied to obtain a valid solution. First, each duplicate is eliminated by removing one of the two copies; then, vertices that were left apart are reinserted in the solution through a simple insertion heuristic. Other related operators where fragments of routes or whole routes are transferred to the offspring are reported in [61, 79, 80]. These operators are applied to a detailed multi-part solution representation, where each part is a string that encodes a route using the classical path representation.

4.4 Decoder

A decoder is a hybrid system where a GA is coupled with a construction heuristic. In a vehicle routing context, insertion heuristics are typically used to add vertices one by one into the current solution. The insertion order is specified by a string of vertices. For example, the string 4-2-3-6-5 means that vertex 4 is the first to be inserted in the solution, followed by vertices 2, 3, 6 and 5, in this order (note that vertex 1 is the depot and does not appear in the string). At any point, a new route can be created to visit a vertex if there is no feasible insertion place in the current set of routes, due to the side constraints. In the case of string 4-2-3-6-5, for example, an individual route is first created for vertex 4. Then, vertices 2 and 3 are inserted in this route. At this point, if we assume that there is no feasible insertion place for vertex 6, a new route is created for this vertex. Finally, vertex 5 is inserted in one of the two routes, depending on its best feasible insertion place. Clearly, different insertion orders are likely to lead to different routes. The GA thus searches for the ordering that will lead to the best solution. This problem-solving approach is attractive

because the GA does not need to explicitly consider side constraints, since they are handled by the insertion heuristic. In a decoder system, specialized crossover and mutation operators are used to produce new offspring orderings from two parent orderings, like those developed for the TSP [62].

We illustrate this idea with a decoder system for the VRPTW reported in [5]. This GA uses two order-based crossover operators *MX1* and *MX2* to produce new insertion orders. Since it is generally desirable to insert vertex i before vertex j in a route if the time window at i occurs before the time window at j, the two operators are strongly biased by this time window-based precedence relationship. In Figure 7, we show how the *MX1* operator produces a valid ordering from two parent orderings.

(a)

parent 1: **2 6 3 4 5**
parent 2: 5 4 2 3 6
offspring: **2** - - - -

(b)

parent 1: **2 6 3 4 5**
parent 2: 2 4 5 3 6
offspring: **2** 4 - - -

(c)

parent 1: **2 4 3 6 5**
parent 2: 2 4 5 3 6
offspring: **2** 4 **3** - -

(d)

parent 1: **2 4 3 6 5**
parent 2: 2 4 3 5 6
offspring: **2** 4 **3** 5 -

(e)

parent 1: **2 4 3 5 6**
parent 2: 2 4 3 5 6
offspring: **2** 4 **3** 5 **6**

Fig. 7. MX1 crossover

This example is based on five vertices (apart from the depot 1), numbered from 2 to 6, and with the following time window precedence relationship $2 < 3 < 4 < 5 < 6$ (i.e., the time window at vertex 2 is the earliest and the time window at vertex 6 is the latest). Starting at position 1, the vertices on the two parent orderings are compared and the vertex with the earliest time window is added to the offspring. Then, the procedure is repeated for the next position, until all positions are done. In Figure 7, vertices 2 and 5 are first compared. Since the time window at vertex 2 occurs before the time window at vertex 5, vertex 2 is selected and takes the first position in offspring (a). To produce a valid ordering, vertices 2 and 5 are then swapped in parent 2. Vertices 3 and 5 are then compared at position 2, and vertex 3 is selected to take the second position in offspring (b). Vertices 5 and 3 are then swapped in parent 1, etc. The resulting offspring is shown in (e). Note that *MX1* pushes vertices with early time windows to the front of the resulting ordering. These vertices are thus the first to be inserted in the solution by the insertion heuristic.

Specialized order-based mutation operators are also used in decoder systems to modify the orderings, for example by moving a vertex at another position or by exchanging the position of two vertices within the ordering. The generality of the decoder approach has led to its application to difficult vehicle routing problems with many side constraints. For example, a successful implementation for a vehicle routing problem with backhauls and time windows is described in [64]. In this work, solutions within 1% of the optimum (on average) have been produced on problem instances with up to 100 vertices.

Before closing this section, we need to mention some results obtained with Evolution Strategies (ES), an evolutionary approach developed in the 60's and 70's at the Technical University of Berlin by Rechenberg and Schwefel (for an introductory text on the subject, see [4]). Like genetic algorithms, ES evolve a population of solutions through mutation and recombination (crossover). One distinctive feature is the concurrent evolution of strategy parameters which are typically properties of the mutation operator, like a mutation step size. There is also a specific population replacement mechanism. For a population of size μ, either the μ best solutions are selected out of λ offspring solutions ($\mu < \lambda$), which is known as a (μ, λ)-selection, or out of the union of the parent and offspring solutions, which is a $(\mu+\lambda)$-selection. The latter scheme is elitist and guarantees a monotonically improving performance. A special case is the (1+1)-ES where there is no recombination operator, only a mutation operator.

Homberger and Gehring [35] were the first to apply ES to the VRPTW using a (μ, λ)-ES. The mutation operator is based on local search heuristics and the mutation step size is the number of modifications to the current solution. An additional strategy parameter associated with the mutation operator specifies the objective to be favored (either minimization of the total distance or minimization of the number of vehicles). Two evolution strategies, called *ES1* and *ES2* are proposed. In the simple variant *ES1*, there is no recombina-

tion, only mutation. Also, the two mutation strategy parameters are directly transferred to the offspring without any modification. This is to be opposed to the second variant *ES2*, where a crossover operator is applied to the mutation step sizes of the parents. In a follow-up paper [36], the authors describe a two-stage approach where $ES1$ is first applied with the objective of minimizing the number of vehicles, followed by a tabu search heuristic to minimize the total distance.

5 Ant Colony Optimization

In their search for food sources, ants initially look around in a random manner. When they find one, they come back to the nest, laying down an aromatic substance on the ground, known as pheromone. The amount of pheromone is related to the quality of the food source, as determined by the quantity of food and its distance from the nest. The next ants will thus search in a less random fashion, as they will be attracted by the pheromone trails. More ants will be attracted on paths with more pheromone, which in turn will lead to more and more pheromone laid down on these paths. Ultimately, all ants will be attracted by the best path.

Ant colony optimization is motivated from the way real ants find good paths to food sources, in particular the indirect communication scheme through pheromone trails that lead to this optimization, a phenomenon called stigmergy. Different ant-based metaphors are reported in the literature, starting from the original Ant System (AS) [16, 18] to more recent variants, like the Ant Colony System (ACS) [17]. Although ant-based systems have first been tested on the TSP, many other combinatorial problems have since been addressed. In the following, the basic AS algorithm for solving the TSP [17] is described and some of its variants are briefly introduced (for a full account, the reader is referred to [19]). Applications for vehicle routing problems are then reviewed.

5.1 Ant System

We assume that we have m ants, where each ant is assigned to a starting location on a vertex of the graph, according to some a priori assignment procedure (which might be random). During one iteration of the method, each ant constructs a tour by repeatedly moving to a new vertex through the application of a probabilistic nearest neighbor transition rule. When all ants have completed their respective tour, the pheromone amount on each edge is modified to favor the edges that are associated with the best tours. This procedure is repeated until a stopping criterion is met, for example a fixed number of iterations.

A simple pseudo-code description for solving the TSP is given below.

1. Assign each ant to a starting vertex;
2. Put some initial amount of pheromone on every edge;
3. For T iterations do:
   3.1 for each ant do: construct a tour;
   3.2 update the amount of pheromone on each edge;
4. Output the best tour found.

In Step 3.1, each ant constructs a complete tour through the repeated addition of a new vertex to the current partial tour. Assuming that ant k is currently located at vertex i, a probabilistic transition rule determines the next vertex j to be visited, namely:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha \ [\eta_{ij}]^\beta}{\sum_{l \in N^k} [\tau_{il}]^\alpha \ [\eta_{il}]^\beta} \quad \text{if } j \in N^k. \tag{1}$$

Among all admissible edges that lead from vertex i to a vertex in set N^k, which is the set of vertices not yet visited by ant k, the probability to select edge (i, j) and go to vertex j is derived from two values: the visibility (closeness) η_{ij} which is the inverse of c_{ij} and τ_{ij} which is the amount of pheromone on edge (i, j). The amount of pheromone on each edge is the memory of the system and depends on how many times that edge was used in the past to construct good tours. Thus, a short edge with a large amount of pheromone has a higher probability of being selected, since it was part of many good tours in previous iterations and it only slightly increases the total distance traveled. In equation (1), the relative influence of the pheromone and visibility information is adjusted via parameters α and β.

Once each ant has constructed a tour, the amount of pheromone is updated in Step 3.2 with the following equation:

$$\tau_{ij} \leftarrow (1 - \rho) \ \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{2}$$

where $0 < \rho \leq 1$ is the pheromone persistence (or, alternatively, $(1 - \rho)$ is the pheromone evaporation). This parameter is used to avoid unlimited accumulation of pheromone on the edges and favors the exploration of new paths. In this equation, $\Delta\tau_{ij}^k$ is the amount of pheromone deposited by ant k on edge (i, j). It is defined as:

$$\Delta\tau_{ij}^k = \begin{cases} 1/C^k & \text{if edge } (i,j) \text{ is visited by ant } k \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where C^k is the total distance traveled by ant k. Thus, a larger amount of pheromone is deposited if the tour is shorter.

A number of refinements and improvements have been proposed over the years to this basic search scheme. They are presented in the following.

5.2 Elitist Ant Systems

A first variant, called the Elitist AS (EAS), reinforces the arcs belonging to the best tour found since the start of the algorithm. Accordingly, equation (2) is modified as follows:

$$\tau_{ij} \leftarrow (1-\rho)\ \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{best} \tag{4}$$

where e is a parameter and $\Delta\tau_{ij}^{best}$ is defined as

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/C^{best} & \text{if edge } (i,j) \text{ is in the best tour} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

with C^{best} the length of the best tour since the start of the algorithm. We thus have e elitist ants that follow the best tour and deposit additional pheromone on it. An alternative approach is the rank-based AS, called AS_{rank}, where the ants are sorted from best to worst according to the length of their tour [8]. The $w-1$ best-ranked ants then deposit an amount of pheromone that is weighted by their rank. More precisely, equation (2) becomes

$$\tau_{ij} \leftarrow (1-\rho)\ \tau_{ij} + \sum_{k=1}^{w-1} (w-k)\ \Delta\tau_{ij}^{k} + w\Delta\tau_{ij}^{best} \tag{6}$$

where k is a rank (from 1 for the best-ranked ant to $w-1$). Note that an additional elitist ant follows the best tour found since the start of the algorithm. This ant deposits an amount of pheromone on the edges weighted by w.

Another approach is the Max-Min AS (MMAS) reported in [77]. Its main characteristic is the exclusive reinforcement of the best tour found at the current iteration or the best tour found since the start of the algorithm. To prevent convergence to suboptimal tours due to the accumulation of pheromone on the same edges, MMAS then forces the amount of pheromone on every edge to lie within an interval $[\tau_{min}, \tau_{max}]$, where τ_{min} and τ_{max} are parameters.

5.3 Ant Colony System

The Ant Colony System (ACS) reported in [17] is more aggressive than AS during the construction of a tour by focusing on the closest vertex when an ant must decide where to go next. That is, the edge leading to the closest vertex is considered first and with high probability. The other edges will only be considered according to equation (1) if this edge is not selected. Also, the best tour found since the start of the algorithm is strongly reinforced. In the so-called global update rule, which corresponds to equation (2) in the AS algorithm, pheromone is deposited or removed only on the edges of the best tour, thus reducing the complexity of this update by an order of magnitude. There is also a local update rule, which is applied each time an ant crosses a

new edge during the construction of its tour. This update significantly reduces the amount of pheromone on this edge and makes it less attractive for the other ants, thus favoring the exploration of new paths.

It is worth noting that improvements are obtained by coupling ant-based systems with local search heuristics. Namely, the solutions constructed by the ants can be further improved through the application of local search heuristics. The pheromone update rules are then based on the value of the local minima.

5.4 Ant-based Systems for Vehicle Routing

Given that vehicle routing problems are a natural extension of the TSP, it is not a surprise if a number of applications of ant-based systems are reported in the literature.

The first work was done in the late '90s on a VRP with maximum distance constraints [6]. In this work, the transition rule of the EAS in equation (4) is modified by forcing the ant to return to the depot and start a new route whenever the inclusion of the next vertex is infeasible due to capacity or maximum distance constraints. Further improvements are then proposed through more substantial modifications to the transition rule. To account for the location of the current vertex i and the next vertex j relative to the depot, a savings term s_{ij} is added to the transition rule. Clearly, the probability to go to vertex j is higher when the savings is larger. To take advantage of the available vehicle capacity, the term $\kappa_{ij} = (Q_i + q_j)/Q$ is also included in the transition rule, where Q_i is the capacity already used, including the demand of vertex i. As this value approaches 1, the vehicle capacity gets more fully exploited. The new transition rule is then:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha \ [\eta_{ij}]^\beta \ [s_{ij}]^\gamma \ [\kappa_{ij}]^\lambda}{\sum_{l \in N^k} [\tau_{il}]^\alpha \ [\eta_{il}]^\beta \ [s_{ij}]^\gamma \ [\kappa_{ij}]^\gamma} \quad \text{if } j \in N^k, \tag{7}$$

where γ and λ are additional parameters to weight the new terms.

In another work by the same authors [7], the visibility term η_{ij} in equation (1) is replaced by a parameterized savings term $s'_{ij} = c_{1i} + c_{1j} - g\ c_{ij} + f\ |c_{1i} - c_{1j}| = s_{ij} - (g-1)\ c_{ij} + f\ |c_{1i} - c_{1j}|$, where f and g are weighting parameters. The authors also replace the EAS framework by AS_{rank}. In both papers, a local search heuristic based on 2-opt arc exchanges is applied to every route in every solution constructed by the ants for further improvement. The results reported on the VRP instances of Christofides, Mingozzi and Toth [10] show that the proposed approach can generate solutions of good quality, but these solutions are not really competitive with state-of-the art metaheuristics based on tabu search.

The idea of incorporating a savings measure to guide the search is explored further in [68]. Here, the authors replace the classical nearest neighbor transition rule by a probabilistic variant of the savings heuristic (see Section 3.1). Here, the attractiveness measure is defined as:

$$\eta_{ij} = [\tau_{ij}]^{\alpha} \ [s_{ij}]^{\beta} \tag{8}$$

where s_{ij} is the savings measure. The largest attractiveness values are then stored in set Ω and the transition rule selects one of them with probability

$$p_{ij}^{k} = \begin{cases} \frac{\eta_{ij}}{\sum_{(h,l)\in\Omega_k} \eta_{hl}} & \text{if } \eta_{ij} \in \Omega \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

After merging the two routes associated with the selected attractiveness value, the number of routes is reduced by one and the procedure is repeated until no further reduction is possible. This probabilistic savings heuristic, embedded within the AS_{rank} framework, is shown to provide significant improvements over the previous results obtained in [6, 7] on the benchmark instances of Christofides, Mingozzi and Toth. Furthermore, the results are now almost at par with the best state-of-the-art methods for the VRP. A further development of this algorithm in [66] is aimed at substantially reducing the computational effort by decomposing the problem into smaller subproblems made of subsets of geographically close routes and by solving each subproblem separately. The authors also consider a local search heuristic based on the swap of two vertices between two routes to further improve the solutions, in addition to the 2-opt arc exchanges. This approach allowed to authors to solve large-scale VRP instances with a few hundred vertices in reasonable computation times.

Ant-based systems have also been applied to the VRPTW. In [24], the proposed multiple ant colony system (MCAS) defines two ant colonies: ACS-VEI that minimizes the number of vehicles (main objective) and ACS-TIME that minimizes the total travel time (secondary objective). Both colonies use the standard ACS framework, except that during the construction of a solution each ant has the choice to go either to a vertex that does not violate the capacity and time window constraints or to return to the depot. ACS-VEI searches for a solution that maximizes the number of visited vertices with a fixed number of vehicles m. When a feasible solution that includes all vertices is obtained, ACS-TIME then constructs solutions with that number of vehicles, while trying to minimize the total travel time. Also, ACS-VEI is restarted with one fewer vehicle. This is repeated until the number of vehicles cannot be reduced anymore. This approach is very similar to the one used by the EA described in Section 4.3 and reported in [2]. The proposed local search heuristic for improving the solutions is based on CROSS exchanges, where two strings of consecutive vertices are exchanged between two routes [78]. Computational experiments on the 56 Euclidean VRPTW benchmark instances of Solomon [76] have produced some best known solutions and have demonstrated that the proposed ant-based system was competitive with the best methodologies at the time.

In [67], the nearest neighbor rule is replaced by a randomized variant of Solomon's $I1$ sequential insertion heuristic [76]. This approach is used to solve the VRPBTW, where the backhauls must follow the linehauls in each route.

Other variants of the VRPBTW, where backhaul and linehaul vertices can be mixed, are studied in [69]. At each iteration of the construction process, the transition rule selects an unvisited vertex i and inserts it at a feasible place between two consecutive vertices j and j' in the current route. Since the procedure involves the replacement of edge (j, j') by two new edges (j, i) and (i, j'), the probability for ant k to select vertex $i \in N^k$ and to insert it between j and j' is:

$$p_i^k = \frac{\max_{j \in R_i}[\eta_{ij} \ \frac{\tau_{ji}+\tau_{ij'}}{2\tau_{jj'}}]}{\sum_{h \in N^k} \max_{l \in R_h}[\eta_{hl} \ \frac{\tau_{lh}+\tau_{hl'}}{2\tau_{ll'}}]}$$

where R_i is the set of vertices after which vertex i can be feasibly inserted and the attractiveness η_{ij} for inserting vertex i just after j is:

$$\eta_{ij} = \max\{0, \alpha c_{1i} - \beta\left(c_{ji} + c_{ij'} - c_{jj'}\right) - (1-\beta)\left(b^i_{j'} - b_{j'}\right)\}$$

In this formula, $b_{j'}$ is the current service time at j' and $b^i_{j'}$ is the new service time after the insertion of vertex i between j and j'. Thus, the attractiveness of vertex i is higher when it is located far from the depot, and when it introduces a small detour and a small service delay in a route. The authors also introduce an additional term in the formula, weighted by parameter γ, to further differentiate between backhaul and linehaul vertices. The transition formula thus accounts for both the attractiveness of vertex i and the amount of pheromone on the edges. In particular, when $(\tau_{ji}+\tau_{ij'})/2\tau_{jj'} > 1$, the average amount of pheromone on the two new edges is larger than the amount of pheromone on the removed edge. This approach, embedded either within the AS_{rank} or ACS framework, was shown to outperform some recent heuristics for the VRPBTW [84, 88] on a data set derived from Solomon's VRPTW instances [26], but it could not match the results of a large neighborhood search [72].

6 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is also a population-based approach where each solution, called a particle, is associated with an objective value and a velocity [40]. The latter defines a direction in the search space, as well as an amplitude, for modifying the solution and is determined through interaction with other solutions. The velocity of particle i at iteration t is obtained through a randomized weighted sum of three components: the velocity of particle i at iteration $t-1$, the direction associated with the best solution achieved by particle i in the past and the direction associated with the best solution achieved over all particles (or over particles that are sufficiently close to particle i, depending on the variant). Although PSO has mostly been used for solving continuous optimization problems, discrete variants also exist.

In [9], a binary version of PSO is used, where a solution or particle is a binary vector. The velocity vector is made of values between 0 and 1 and the corresponding solution is obtained by interpreting each velocity value as the probability that the corresponding element in the solution is 0 (and, conversely, as 1 minus the probability that the corresponding element is 1). This approach is exploited to solve the VRP. Basically, a particle is a vector of size $n \times m$, where n is the number of vertices and m is the number of vehicles. That is, the vector of vertices is duplicated m times, one time for each vehicle. An entry of 1 means that the vertex is visited by the corresponding vehicle. The PSO thus solves the assignment subproblem. As the solution obtained is not necessarily valid (i.e., exactly one position associated with a vertex should be equal to 1), a repair operator is used. If the solution is valid but infeasible, the solution is discarded and the velocity is recalculated until a feasible solution is obtained. A simulated annealing heuristic is then used to sequence the vertices in each route.

In [89], an alternative particle representation with only $n+m-1$ elements is used for solving the VRPTW. Each entry corresponds to the position of one of the n vertices or one of the $m-1$ copies of the depot in the solution vector. The PSO thus solves the assignment and sequencing problems concurrently. The objective is the total distance plus penalties for violations of the capacity and time constraints. The velocity is a vector of values between -$(n+m-2)$ and $n+m-2$ that is added to the solution vector to update it. Normalization is then applied to obtain new integer positions.

7 Neural Networks

Neural networks (NN) are composed of units that perform in a manner similar to neurons in the human brain. These units are richly interconnected through weighted connections: a signal is sent from one unit to another along a connection and is modulated by its associated weight (which stands for the strength or efficiency of the connection). Although superficially related to their biological counterpart, NNs posses characteristics associated with human cognition. In particular, they can learn from experience and induce general concepts from specific examples. When presented with a set of inputs, they self-adjust to produce the desired response.

Neural networks were originally designed for tasks well adapted to human intelligence and where traditional computation was inadequate, such as speech understanding, artificial vision and handwritten character recognition. Starting with the pioneering work of Hopfield and Tank [37] on the TSP, they have also been applied to combinatorial optimization problems. Although neural network models can handle spatial relationships among vertices, they cannot easily handle side constraints, such as capacity constraints or time windows, that prevent a pure spatial interpretation of the problem. Therefore, the literature on NNs applications to vehicle routing problems is scant, except for

a special class of models, known as deformable templates, like the elastic net and the self-organizing map [29, 28, 50, 55, 75, 86]. The work of Ghaziri will be used in the following to illustrate how a deformable template, here a self-organizing map, can be applied to the standard VRP with capacity constraints.

7.1 Self-Organizing Maps

Self-organizing maps [42] are composed of a layer of input units fully connected to a layer of output units, where the output units are organized into a particular topology, such as a ring. These models self-organize through an iterative adjustment of their connection weights to find some regularity or structure in the input data. They are typically used to categorize or cluster data. In Figure 8, T_{11} and T_{21} denote the weights on the connections from the two input units I_1 and I_2 to output unit 1, and $T_1 = (T_{11}, T_{21})$ is the weight vector associated with output unit 1.

Assuming an ordered set of n input vectors of dimension p and a self-organizing map with p input units and q output units on a ring, the basic algorithm for adjusting the connection weights is the following.

1. *Initialization.* Set each connection weight to some initial value.
2. *Competition.* Consider the next input vector I. If all input vectors are done, restart from the first input vector. Compute the output value o_j of each output unit j as the weighted sum of its inputs, that is, $o_j = \sum_{i=1}^{p} T_{ij}\ I_i,\ j = 1, ..., q$, where T_{ij} is the connection weight between input unit i and output unit j. The winning output unit $j*$ is the unit with maximum output value.
3. *Weight adjustment.* Modify the connection weights of each output unit by setting $T_j \leftarrow T_j + f(j, j*)\,(I - T_j)$, where $T_j = (T_{1j}, T_{2j}, ..., T_{pj})$ is the weight vector of output unit j, and f is a function of j and $j*$.
4. Repeat Steps 2 and 3, until the weight vectors stabilize. At the end, each input vector is assigned to the output unit that maximizes the weighted sum in Step 2.

In Step 3, f is typically a decreasing function of the lateral distance between units j and $j*$ on the ring (i.e., if there are k units on the ring between the two units, the lateral distance is $k + 1$) and its range is the interval [0,1]. Thus, the weight vector of the winning unit $j*$ and the weight vectors of units that are close to $j*$ on the ring all move toward the input vector I, but with decreasing intensity as the lateral distance to the winning unit increases. Typically, function f is modified as the learning algorithm unfolds to gradually reduce the magnitude of the weight adjustment. At the start, all units close to the winning unit on the ring follow that unit in order to cover a particular area. At the end, only the weight vector of the winning unit significantly moves toward the current input vector, to fix the assignment.

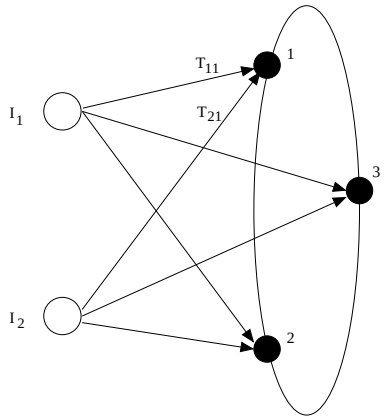

Fig. 8. Self-organizing map

Self-organizing maps produce topological mappings from high-dimensional input spaces to low-dimensional output spaces. In the case of a ring, each p-dimensional input vector is associated with a 1-dimensional position on the ring. The mapping is such that two input vectors that are close in the input space are assigned to units that are close on the ring.

7.2 Self-Organizing Maps for Vehicle Routing

In a VRP context, an input vector is made of the (x,y) coordinates of a vertex and a different ring is associated with each vehicle route. The learning algorithm is also slightly modified. Assuming an Euclidean distance metric, the winning output unit is the one with the closest weight vector to the coordinate vector of the current vertex (rather than the one with the largest correlation with the input vector, as obtained by maximizing the inner product, see Step 2 of the basic algorithm). Thus, the weight vector of any given output unit can be interpreted as the location of this unit in the Euclidean space. Accordingly, the procedure stops when there is a unit (weight vector) sufficiently close to each vertex. Given that multiple rings are used, the winning unit must be chosen among all units over all rings. Then, the units located on the ring of the winning unit move toward the current vertex. At the end of this procedure, each vertex is assigned to its closest unit. Through this assignment, the ordering of the units on each ring defines an ordering of the vertices on each route. Furthermore, since close units on a given ring tend to migrate toward neighboring vertices in the Euclidean plane, short routes are expected to emerge. Figure 9 depicts the evolution of the rings in a two-dimensional Euclidean plane, starting from the initial configuration illustrated in Figure 9(a).

In this figure, the vertices are the white nodes and the units are the small black nodes. Note that the location of each unit corresponds to its current

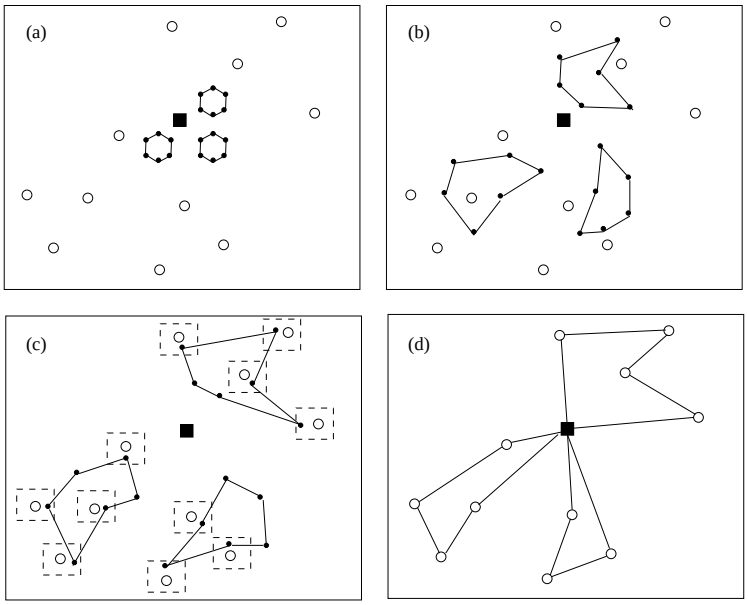

Fig. 9. Ring migration

weight vector. At the start, no solution of the problem can be inferred. However, as the weight vectors migrate toward the vertices, the solution progressively emerges. The final solution is determined at the end, during the final assignment of each vertex to a unit.

This type of approach is now illustrated with the so-called Hierarchical Deformable Net [29]. The model involves a competition at two different levels, namely, a deterministic competition among the units within each ring, and a stochastic competition among the rings. The number of rings is fixed, but the model is applied with an increasing number of rings (routes) until a feasible solution is obtained. In the following, n is the number of vertices, m is the number of rings, $d(x, y)$ is the Euclidean distance between the two-dimensional vectors x and y and $dL(j, l, k)$ is the lateral distance between units j and l on ring r^k, $k = 1, ..., m$.

1. Randomly order the vertices and create one ring per route. Each ring contains $2\lceil(n/m)\rceil$ units. Set $h \leftarrow 1$ and $G \leftarrow 4\lceil(n/m)\rceil$.
2. Repeat Step 3 to Step 8 until there is a weight vector sufficiently close to each vertex.
3. Select the next vertex i and set the current input vector I to its coordinates (if all vertices are done, restart with the first vertex).
4. Competition within each ring. Determine the winning unit o^k_{j*} on each ring r^k. This unit is the one with the closest weight vector to the current input vector I.

5. Competition among the rings. Assign the following winning probability to each ring r^k:

$$p\left(r^k\right) = \frac{f\left(r^k, h\right)}{\sum_{k'=1,\ldots,m} f\left(r^{k'}, h\right)} \tag{10}$$

$$f\left(r^k, h\right) = \frac{e^{-d\left(T^k_{j*}, I\right)/h}}{1 + e^{-\Delta Q/h}} \tag{11}$$

where ΔQ is the difference between the vehicle capacity and the total demand on ring r^k (with current vertex i).
6. Select the winning ring r^{k*}, using a selection probability for each ring r^k proportional to $p\left(r^k\right)$. Vertex i is assigned to unit o^{k*}_{j*}.
7. Move the weight vector T^{k*}_j of each unit j on the winning ring r^{k*} toward the coordinate vector I of vertex i:

$$T^{k*}_j \leftarrow T^{k*}_j + f\left(j, j*, k*\right)\left(I - T^{k*}_j\right) \tag{12}$$

$$f\left(j, j*, k*\right) = \frac{e^{-dL(j,j*,k*)^2/G^2}}{\sqrt{2}} \tag{13}$$

8. Slightly decrease the value of parameters h and G, by setting $h \leftarrow 0.99\ h$ and $G \leftarrow 0.9\ G$.

In Step 5, parameter h is used to modify the probability distribution over the rings. When the value of h is high, the selection probability of each ring is about the same. When the value of h is small, the probability distribution favors the ring that is closest to the current vertex among all rings that satisfy the capacity constraint (with the current vertex). In Step 7, parameter G controls the magnitude of the moves. When its value is reduced, the magnitude is also reduced. Hence, the values of both parameters h and G are gradually lowered to favor convergence toward good feasible solutions.

Ghaziri has applied this model to a few VRPs taken from the data set of Christofides, Mingozzi and Toth [10]. Although the solutions produced were of relatively good quality, they were not competitive with the best solutions produced with alternate problem-solving methods, like tabu search. In a later study [28], this model is extended to solve the VRP with maximum route time constraints. In this case, the selection probabilities are adjusted through an additional component that accounts for these constraints. As opposed to capacity constraints, however, the satisfaction of maximum route time constraints depends on the sequence of vertices within each route. At each iteration, a TSP procedure is thus applied to the currently assigned vertices on each ring (including the current vertex) to identify a plausible sequence. This sequence is then used to compute the selection probabilities.

Neural networks have not led to a steady stream of research for solving combinatorial optimization problems. In the case of vehicle routing, we do not

know of any recent significant contribution in this field, with the exception of [30] where a self-organizing map for the VRPB is proposed. This approach is shown to be competitive with the best know methods on small sized instances, but its performance tend to deteriorate with increasing sizes.

8 Artificial Immune Systems

AIS is a recent bio-inspired algorithm motivated from the mechanisms used by our organism to defend itself against foreign microorganisms, like viruses and bacteria [13, 15]. In the natural immune system, molecular patterns at the surface of microorganism, known as antigens, lead to the proliferation of lymphocytes that are able to produce antigen-specific antibodies. This phenomenon is known as clonal selection (or clonal expansion). Since each lymphocyte has a distinct antigen specificity and the organism needs to react to different antigens, the number of lymphocytes that can differentiate into effector cells able to produce antibodies for each specific antigen is limited. Thus, it is primordial for the organism to develop a good repertoire of lymphocytes with different specificities. Furthermore, the differentiating process is not of the "all-or-nothing" type, so that antibodies typically show a variable degree of antigen affinity or specificity.

In AIS developed for optimization purposes, the antigen affinity corresponds to the solution value. Then, different mechanisms observed in the immune system, like the clonal selection mechanism, are exploited to generate good solutions. For example, artificial clonal selection involves the selection of solutions with the best objective values (high affinity), the reproduction of these solutions to obtain a number of copies or clones and mutation of these clones to generate improved solutions. As opposed to classical genetic algorithms, where mutation is a random process which is applied at low rates, the mutation operator in AIS is controlled and its rate is inversely proportional to the solution value. It is also typically applied at high rates (hypermutation) to increase population diversity. These AIS principles are often integrated within GAs to allow a better management of the population of solutions.

This is the case for example in [39], where an additional operator motivated from AIS is applied after crossover to allow the proliferation of high quality solutions in the population. This approach is applied for solving an electricity distribution network problem. The problem is modeled as a multiple depot vehicle routing problem, where the depots correspond to high/medium voltage stations and the other vertices to medium/low voltage stations. In [49], an immune-based operator is used within a GA to solve the VRPTW. Basically, the operator is able to recognize good patterns in solutions, such as subpaths that link close vertices. It then uses this information to improve solutions. For example, if vertex i is close to vertex j and the two vertices are not in the same route or are in the same route but are not consecutive, then this operator will put j just after i if an improvement is obtained.

Since the field of AIS is still in its infancy, there is only a few work related to combinatorial optimization and vehicle routing. However, this line of research should expand in the next few years with the availability of a number of paradigms to model immunological processes, like the immune network models [23].

9 Hybrids

An increasing number of bio-inspired algorithms do not rely on a single search strategy but rather combine various algorithmic ideas. These approaches are referred to as hybrids. The motivation behind hybrids is usually to exploit the strengths of a number of individual search strategies to obtain a more powerful problem-solving approach. We have already mentioned some hybrid algorithms in the preceding sections. For example, a decoder system can be viewed as the hybrid of a GA and a construction heuristic. The integration of local search-based mutation operators within GAs, known as memetic algorithms [56], is another type of hybrid. Likewise, some AIS concepts have been integrated within GAs to make them more robust.

In this regard, two recent developments for solving the VRP and VRPTW need to be mentioned. In [46, 47], different tabu searches and GAs are used to explore the search space concurrently. Their interaction is realized through a central warehouse, which is basically an adaptive memory made of elite solutions [71]. The diversity of solutions in the warehouse is exploited when new starting solutions are provided to the tabu searches and GAs. With this approach, the authors have found solutions that are competitive with state-of-the-art methods on Solomon's VRPTW instances [76], as well as on the extended data set of Gehring and Homberger [25].

In [51], an algorithm called AGES combines guided local search (GLS) [87] and ES in an iterative two-stage procedure to address both the VRP and VRPTW. GLS introduces modifications into the objective function through penalties when the search gets trapped in a local optimum. Here, the penalty counter of one arc in the solution is incremented by one, each time a local optimum is reached. The arc is selected on the basis of its length (a long arc is more likely to be penalized) and current penalty counter (an arc with a high penalty counter is less likely to be penalized again). In the first stage, GLS controls the objective function of a composite local search that exploits different types of modifications to the solution, while in the second stage, it controls the objective function of a $(1 + 1)$-ES. In this ES, the ruin and recreate principle is at the core of the mutation operator. That is, a number of vertices are removed from the current solution and reinserted at least cost. Refinements to this algorithm are proposed in [52, 53] with excellent results reported on classical VRP and VRPTW benchmark instances. On the VRP data sets of Christofides, Mingozzi and Toth [10] and Golden et al. [33], in

particular, this approach has produced the best average solution values when compared with state-of-the-art methods.

10 Conclusion

This chapter has reviewed the main bio-inspired algorithms that have been used to solve vehicle routing problems. From this review, it is clear that evolutionary algorithms and their hybrids have shown the best performance up to now. They have been competitive and have even outperformed widely used metaheuristics, like tabu search, on some classical problems like the VRP and VRPTW. At the other end of the spectrum, neural networks have been more or less abandoned due to their deceiving performance on standard computer platforms. It is possible that the availability of specialized hardware dedicated to neural network implementations will allow researchers to exploit their full power, thus leading to their resurgence in vehicle routing studies. In-between stand ant colony optimization with some interesting results. Finally, particle swarm optimization and artificial immune systems are not yet mature fields of research. Consequently, only scarce results are reported up to now, but these approaches certainly show great promises for the future.

Acknowledgments. Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC). This support is gratefully acknowledged.

References

1. Ahuja RK, Ergun O, Orlin JB, Punnen AP (2002) A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics 123:75–102
2. Berger J, Barkaoui M, Bräysy O (2003) A route-directed hybrid genetic approach for the vehicle routing problem with time windows. INFOR 41:179–194
3. Berger J, Salois M, Bégin R (1998) A hybrid genetic algorithm for the vehicle routing problem with time windows. In: Mercer RE, Neufeld E (eds) Lecture Notes in Computer Science 1418. Springer, London 114–127
4. Beyer H-G, Schwefel H-P (2002) Evolution strategies: A comprehensive introduction. Natural Computing 1:3–52
5. Blanton JL, Wainwright RL (1993) Multiple vehicle routing with time and capacity constraints using genetic algorithms. In: Forrest S (ed) Proceedings of the 5th International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA 451–459
6. Bullnheimer B, Hartl RF, Strauss C (1999) Applying the ant system to the vehicle routing problem. In: Voss S, Martello S, Osman IH, Roucairol C (eds) Meta-heuristics: Advances and trends in local search paradigms for optimization. Kluwer, Boston, MA 285–296

7. Bullnheimer B, Hartl RF, Strauss C (1999) An improved ant system algorithm for the vehicle routing problem. Annals of Operations Research 89:319–328
8. Bullnheimer B, Hartl RF, Strauss C (1999) A new rank-based version of the ant system: A computational study. Central European Journal for Operations Research and Economics 7:25–38
9. Chen A-L, Yang GK, Wu ZM (2006) Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem. Journal of Zhejiang University SCIENCE A 7:607–614
10. Christofides N, Mingozzi A, Toth P (1979) The vehicle routing problem. In: Christofides N, Mingozzi A, Toth P, Sandi C (eds) Combinatorial Optimization. Wiley, Chichester 315–338
11. Clarke G, Wright J (1964) Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12:568–581
12. Cordeau J-F, Laporte G, Mercier A (2001) A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational Research Society 52:928–936
13. Dasgupta D (ed) (1999) Artificial immune systems and their applications. Springer, Berlin
14. de Castro, LN (2006) Fundamentals of natural computing: Basic concepts, algorithms, and applications. CRC Press, Boca Raton, FL
15. de Castro LN, Timmis J (2002) Artificial immune systems: A new computational intelligence approach. Springer, London
16. Dorigo M (1992) Optimization, learning and natural algorithms. Ph.D. Dissertation, Politecnico di Milano, Italy (in Italian)
17. Dorigo M, Gambardella LM (1997) Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1:53–66
18. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man and Cybernetics 26:29–41
19. Dorigo M, Stützle T (2004) Ant colony optimization, The MIT Press, Cambridge, MA
20. Dror M (eds) (2000) Arc routing: Theory, Solutions and Approximations. Springer, Berlin
21. Eiselt HA, Gendreau M, Laporte G (1995) Arc routing problems, Part 1: The chinese postman problem. Operations Research 43:231–242
22. Eiselt HA, Gendreau M, Laporte G (1995) Arc routing problems, Part 2: The rural postman problem. Operations Research 43:399–414
23. Farmer JD, Kauffman S, Packard NH, Perelson AS (1987) Adaptive dynamic networks as models for the immune system and autocatalytic sets. Annals of the New York Academy of Sciences 504:118–131
24. Gambardella LM, Taillard ÉD, Agazzi G (1999) MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In: Corne D, Dorigo M, Glover F (eds) New ideas in optimization. McGraw-Hill, London 63–76
25. Gehring H, Homberger J (1999) A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In: Miettinen K, Mäkelä MM, Toivanen J (eds) Proceedings of EUROGEN99 - Short course on evolutionary algorithms in engineering and computer science. Reports of the Department of

Mathematical Information Technology, No. A 2/1999, University of Jyväskylä, Finland 57–64
26. Gelinas S, Desrochers M, Desrosiers J, Solomon MM (1995) A new branching strategy for time constrained routing problems with application to backhauling. Annals of Operations Research 61:91–109
27. Gendreau M, Laporte G, Potvin J-Y (2002) Metaheuristics for the Capacitated VRP. In: Toth P, Vigo D (eds) The vehicle routing problem. SIAM, Philadelphia, PA 129–154
28. Ghaziri H (1996) Supervision in the self-organizing feature map: Application to the vehicle routing problem. In: Osman IH, Kelly JP (eds) Meta-heuristics: Theory & applications. Kluwer, Boston, MA 651–660.
29. Ghaziri H (1991) Solving routing problems by a self-organizing map. In: Kohonen T, Makisara K, Simula O, Kangas J (eds) Artificial neural networks. North-Holland, Amsterdam 829–834
30. Ghaziri H, Osman IH (2006) Self-organizing feature maps for the vehicle routing problem with backhauls. Journal of Scheduling 9:97–114
31. Gillett B, Miller L (1974) A Heuristic Algorithm for the Vehicle Dispatch Problem. Operations Research 22:340–379.
32. Glover F, Laguna M (1997) Tabu search. Kluwer, Boston, MA
33. Golden BL, Wasil EA, Kelly JP, Chao I-M. (1998) The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets and computational results. In: Crainic TG, Laporte G (eds) Fleet Management and Logistics. Kluwer, Norwell, MA 33–56
34. Holland JH (1975). Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, MI (reprinted in 1992 by The MIT Press, Cambridge, MA)
35. Homberger J, Gehring H (1999) Two evolutionary metaheuristics for the vehicle routing problem with time windows. INFOR 37:297–318
36. Homberger J, Gehring H (2005) A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. European Journal of Operational Research 162:220–238
37. Hopfield JJ, Tank DW (1985) Neural computation of decisions in optimization problems. Biological Cybernetics 52:141–152
38. Jung S, Moon B-R (2002) A hybrid genetic algorithm for the vehicle routing problem with time windows. In: Langdon WB et al. (eds) Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufmann, San Francisco, CA 1309–1316
39. Keko H, Skok M, Skrlec D (2004) Solving the distribution network routing problem with artificial immune systems. In: Proceedings of the IEEE Mediterranean Electrotechnical Conference. Dubrovnik, Croatia 959–962
40. Kennedy J, Eberhart RC (2001) Swarm intelligence. Morgan Kaufmann, San Francisco, CA
41. Kirkpatrick S, Gelatt Jr CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
42. Kohonen T (1988) Self-organization and associative memory. Springer, Berlin
43. Laporte G (1992) The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research 59:345–358
44. Laporte G (1997) Vehicle routing. In: Dell'Amico M, Maffioli F, Martello S (eds) Annotated bibliographies in combinatorial optimization. Wiley, Chichester 223–240

45. Lawler EL, Lenstra JK, Rinnooy Kan AHG, Schmoys DB (eds) (1985) The traveling salesman problem, Wiley, Chichester
46. Le Bouthillier A, Crainic TG (2005) A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. Computers & Operations Research 32:1685–1708
47. Le Bouthillier A, Crainic TG, Kropf P (2005) A guided cooperative search for the vehicle routing problem with time windows. IEEE Intelligent Systems 20:36–42
48. Lin S (1965) Computer solutions of the traveling salesman problem. Bell System Technical Journal 44:2245–2269
49. Ma J, Zou H, Gao L-Q, Li D (2006) Immune genetic algorithm for vehicle routing problem with time windows. In: Proceedings of the Fifth International Conference on Machine Learning and Cybernetics. Dalian, China 3465–3469
50. Matsuyama Y (1991) Self-organization via competition, cooperation and categorization applied to extended vehicle routing problems. In: Proceedings of the International Joint Conference on Neural Networks. Seattle, WA I–385–390.
51. Mester D, Bräysy O (2005) Active guided evolution strategies for large-scale vehicle routing problems with time windows. Computers & Operations Research 32:1593–1614
52. Mester D, Bräysy O (2007) Active guided evolution strategies for large-scale capacitated vehicle routing problems. Computers & Operations Research 34:2964–2975
53. Mester D, Bräysy O, Dullaert W (2007) A multi-parametric evolution strategies algorithm for vehicle routing problems. Expert Systems with Applications 32:508–517
54. Mladenović N, Hansen P (1997) Variable neighborhood search. Computers & Operations Research 24:1097–1100
55. Modares A, Somhom S, Enkawa T (1999) A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems. International Transactions in Operations Research 6:591–606
56. Moscato P, Cotta C (2003) A gentle introduction to memetic algorithms. In: Glover F, Kochenberger GA (eds) Handbook of Metaheuristics. Kluwer, Boston 105–144
57. Nagata Y (2007) Edge assembly crossover for the capacitated vehicle routing problem. In: Cotta C, van Hemert J (eds) Lecture Notes in Computer Science 4446. Springer, Berlin 142–153
58. Nagata Y (2007) Effective memetic algorithm for the vehicle routing problem with time windows: Edge assembly crossover for the VRPTW. In: Proceedings of the Seventh Metaheuristics International Conference, Montreal, Canada (on CD-ROM)
59. Oliver IM, Smith DJ, Holland JRC (1987) A study of permutation crossover operators on the traveling salesman problem. In: Grefenstette JJ (ed) Proceedings of the Second International Conference on Genetic Algorithms and Their Applications. Lawrence Erlbaum Associates, Hillsdale, NJ 224–230
60. Osman IH (1993) Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research 41:421–451
61. Pereira FB, Tavares J, Machado P, Costa E (2002) GVR: A new genetic representation for the vehicle routing problem. In: O'Neill M et al. (eds) Lecture Notes in Computer Science 2464. Springer, Berlin 95–102

62. Potvin J-Y (1996) Genetic algorithms for the traveling salesman problem. Annals of Operations Research 63:339–370
63. Potvin J-Y, Bengio S (1996) The vehicle routing problem with time windows - Part II: Genetic search. INFORMS Journal on Computing 8:165-172
64. Potvin J-Y, Duhamel C, Guertin F (1996) A genetic algorithm for vehicle routing with backhauling. Applied Intelligence 6:345-355
65. Prins C (2004) A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & Operations Research 31:1985–2002
66. Reimann M, Doerner K, Hartl RF (2004) D-Ants: Savings based ants divide and conquer the vehicle routing problem. Computers & Operations Research 31:563–591
67. Reimann M, Doerner K, Hartl RF (2002) Insertion based ants for vehicle routing problems with backhauls and time windows. In: Dorigo M, Di Caro G, Sampels M (eds) From ant colonies to artificial ants: Third international workshop on ant algorithms. Lecture Notes in Computer Science 2463. Springer, Berlin 135–147
68. Reimann M, Stummer M, Doerner K (2002) A savings-based ant system for the vehicle routing problem. In: Langdon WB et al. (eds) Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufmann, San Francisco, CA 1317–1325
69. Reimann M, Ulrich H (2006) Comparing backhauling strategies in vehicle routing using ant colony optimization. Central European Journal of Operations Research 14:105–123
70. Reinelt G (1991) TSPLIB - A traveling salesman problem library. ORSA Journal on Computing 3:376–384
71. Rochat Y, Taillard ÉD (1995) Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1:147–167
72. Ropke S, Pisinger D (2006) A unified heuristic for a large class of vehicle routing problems with backhauls. European Journal of Operational Research 171:750–775
73. Schrimpf G, Schneider J, Stamm-Wilmbrandt H, Duek G. (2000) Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics 159:139–171
74. Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: Maher M, Puget J-F (eds) Lecture Notes in Computer Science 1520. Springer, New-York, NY 417–431
75. Schumann M, Retzko R (1995) Self-organizing maps for vehicle routing problems: Minimizing an explicit cost function. In: Fogelman-Soulie F (ed) Proceedings of the International Conference on Artificial Neural Networks. EC2&Cie, Paris II–401–406
76. Solomon MM (1987) Time window constrained routing and scheduling problems. Operations Research 35:254–265
77. Stützle T, Hoos HH (2000) MAX-MIN ant system. Future Generation Computer Systems Journal 16:889–914
78. Taillard ÉD, Badeau P, Gendreau M, Guertin F, Potvin J-Y (1997) A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science 31:170–186
79. Tavares J, Pereira FB, Machado P, Costa E (2002). GVR delivers it on time. In: Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning. Singapore 745–749

80. Tavares J, Pereira FB, Machado P, Costa E (2003). On the influence of GVR in vehicle routing. In: Proceedings of the ACM Symposium on Applied Computing. Melbourne, FL 753–758
81. Thangiah SR (1995) An adaptive clustering method using a geometric shape for vehicle routing problems with time windows. In: Eshelman LJ (ed) Proceedings of the 6th International Conference on Genetic Algorithms. Morgan Kaufmann, San Francisco, CA 536–543
82. Thangiah SR, Nygard KE, Juell PL (1991) GIDEON: A genetic algorithm system for vehicle routing with time windows. In: Proceedings of 7th IEEE Conference on Artificial Intelligence Applications. IEEE Computer Society Press, Los Alamitos, CA 322–328
83. Thangiah SR, Salhi S (2001) Genetic clustering: An adaptive heuristic for the multidepot vehicle routing problem. Applied Artificial Intelligence 15:361–383
84. Thangiah SR, Potvin J-Y, Sun T (1996) Heuristic approaches to vehicle routing with backhauls and time windows. Computers & Operations Research 23:1043–1057
85. Toth P, Vigo D (eds) (2001) The vehicle routing problem. SIAM, Philadelphia, PA
86. Vakhutinsky AI, Golden BL (1994) Solving vehicle routing problems using elastic net. In: Proceedings of the IEEE International Conference on Neural Networks. IEEE Press, Piscataway NJ 4535–4540
87. Voudouris C, Tsang EPK (1995) Guided local search. Technical Report CSM-247, Department of Computer Science, University of Essex, Colchester, UK
88. Zhong Y, Cole MH (2005) A vehicle routing problem with backhauls and time windows: A guided local search solution. Transportation Research E41:131–144
89. Zhu Q, Qian L, Li Y, Zhu S (2006) An improved particle swarm optimization algorithm for vehicle routing problem with time windows. In: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE Press, Piscataway, NJ 1386–1390