Antoine Kalmbach

Evolutionary Algorithms in Vehicle Routing

A Bachelor’s Thesis
in Information Technology
May 18, 2011

UNIVERSITY OF JYVÄSKYLÄ
DEPARTMENT OF MATHEMATICAL INFORMATION TECHNOLOGY
Jyväskylä
Author: Antoine Kalmbach
Contact information: antoine.kalmbach@jyu.fi
Title: Evolutionary Algorithms in Vehicle Routing
Työn nimi: Evoluutionääriset algoritmit ja kuljetusongelma
Project: A Bachelor’s Thesis in Information Technology
Page count: 32

Abstract: The vehicle routing problem (VRP) is an important problem in operations research. VRP has numerous practical applications in logistics and transportation, and is academically significant for its complexity. In this thesis we study solving VRP using evolutionary algorithms. Evolutionary algorithms are metaheuristics that apply artificial evolution in problem-solving. A new hybrid type of evolutionary algorithms, memetic algorithms, are a recent advance in evolutionary algorithms and their usage in VRP has not been widely studied. As such, this thesis attempts to review recent advances in hybrid evolutionary algorithms in VRP. We conducted a survey of six different memetic algorithms in VRP. The reviewed memetic algorithms could outperform state-of-the-art metaheuristics, and in some cases improved the current best-known results. This would imply that memetic algorithms have great potential in VRP.


Keywords: metaheuristics, vehicle routing problem, genetic algorithms, evolutionary algorithms, memetic algorithms

Avainsanat: metaheuristiikat, kuljetusongelma, geneettiset algoritmit, evoluutioalgoritmit, memeettiset algoritmit
**List of Abbreviations**

CVRP Capacitated Vehicle Routing Problem

DCVRP Distance Constrained Vehicle Routing Problem

DE Differential Evolution

EAMA Edge-Assembly Memetic Algorithm

EAX Edge-Assembly Crossover

EA Evolutionary Algorithm

ES Evolutionary strategy

GA Genetic Algorithm

HFVRP Heterogeneous Fleet Vehicle Routing Problem

LOX Linear Order Crossover

LS Local Search

MA Memetic Algorithm

MA|PM Memetic Algorithm with Population Management

OX Order Crossover

RAR Relocate and Reinsert

TSP Traveling Salesman Problem

VFMP Vehicle Fleet Management Problem

VRPPD Vehicle Routing Problem with Pickup and Delivery

VRPTW Vehicle Routing Problem with Time Windows

VRP Vehicle Routing Problem
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Abbreviations</td>
<td>i</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>2 The Vehicle Routing Problem</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>3 Strategies for Vehicle Routing Problems</strong></td>
<td>4</td>
</tr>
<tr>
<td>3.1 Exact Methods and Classical Heuristics</td>
<td>5</td>
</tr>
<tr>
<td>3.2 Metaheuristics</td>
<td>7</td>
</tr>
<tr>
<td><strong>4 Evolutionary Algorithms</strong></td>
<td>11</td>
</tr>
<tr>
<td>4.1 Background and Principles</td>
<td>11</td>
</tr>
<tr>
<td>4.2 Genetic Algorithms</td>
<td>12</td>
</tr>
<tr>
<td>4.3 Evolutionary Strategies</td>
<td>13</td>
</tr>
<tr>
<td>4.4 Adding Local Improvement</td>
<td>15</td>
</tr>
<tr>
<td>4.5 Memetic Algorithms</td>
<td>16</td>
</tr>
<tr>
<td><strong>5 Genetic and Memetic Algorithms in VRP</strong></td>
<td>17</td>
</tr>
<tr>
<td>5.1 Encoding Routing Problems</td>
<td>18</td>
</tr>
<tr>
<td>5.2 Recombination Procedures</td>
<td>19</td>
</tr>
<tr>
<td>5.3 Incorporating Memes</td>
<td>21</td>
</tr>
<tr>
<td>5.4 Recent Results in Memetic Algorithms</td>
<td>22</td>
</tr>
<tr>
<td><strong>6 Conclusion and Further Research</strong></td>
<td>23</td>
</tr>
<tr>
<td><strong>References</strong></td>
<td>24</td>
</tr>
</tbody>
</table>
1 Introduction

The vehicle routing problem (VRP) is a combinatorial optimisation problem that has been widely studied in operations research. While it is academically challenging due to its complex nature, VRP also has tremendous practical and economical importance in the fields of logistics and transportation. It is especially important due to its applicability, as VRP and its variations are based on tangible real-life problems.

The vehicle routing problem was first described by Dantzig and Ramser [9] as an optimisation problem in which the objective is to find a set of minimum-cost vehicle routes for delivering goods to customers from a central depot. Each customer must be visited exactly once by a single vehicle. The problem has multiple variations. A few of them are the vehicle routing problem with time windows (VRPTW) and the capacitated vehicle routing problem (CVRP). VRPTW adds a time window constraints so that each customer must be served within a given time interval, and CVRP imposes a maximum capacity for each vehicle.

The vehicle routing problem is a NP-hard problem, making it difficult to solve to optimality [39]. While exact algorithms have been developed for VRP, the complexity of VRP calls for the use of heuristics and metaheuristics. Heuristics try to provide near-optimal solutions in a reasonable amount of time instead of doing an exhaustive search and metaheuristics are techniques that control or guide these heuristics in their search, thus operating a level higher than regular heuristics. In VRP, most metaheuristics can be classified as single-solution metaheuristics, in that they optimise a single solution iteratively [54]. Population-based metaheuristics on the other hand optimise the problem by looking at multiple solutions. There are multiple types of population-based algorithms and methodologies, notably genetic algorithms and evolutionary strategies. These methods are called evolutionary algorithms (EA), using techniques adapted from real life, e.g., mutation and reproduction. Evolutionary algorithms seek to produce—to evolve—a group of solutions into better ones. These evolutionary algorithms have been applied to VRP with competitive results [33, 49].

The vehicle routing problem has also seen an emergent use of hybrid evolutionary algorithms, in which evolutionary algorithms are combined with heuristics, for example a local search. Such hybrid EAs were named the memetic algorithm (MA) by [36]. Memetic algorithms for VRP have been produced in [39, 41, 49, 50] yielding high-quality results. Memetic algorithms in VRP are new, and due to their nascent nature, their performance in VRP has not been widely studied. This study attempts to review the recent advances in memetic algorithms in VRP.
The remainder of this thesis is structured as follows. We begin first by describing VRP and providing a formal definition to it in Section 2. Different kinds of algorithms, heuristics and metaheuristics and their use in VRP are then studied in Section 3. In Section 4 we shift focus towards evolutionary algorithms, notably genetic algorithms and evolutionary strategies and provide a general overview of their functionality. We then examine how to combine these different approaches in practice in Section 5. The study concludes with an overview of the status of MA in VRP and presents some possible topics for further research in Section 6.

2 The Vehicle Routing Problem

In this section, we introduce the reader to VRP with the problem formulation. We also present a mixed integer programming model and discuss some of the most significant variants of VRP.

In the vehicle routing problem, the objective is to produce a set of minimum-cost routes for a fleet of vehicles. The vehicles are to deliver goods to customers from a central depot. Each customer must be visited exactly once. The famous Travelling Salesman Problem (TSP) is a variant of VRP insofar as there is no depot and only one vehicle. Figure 1 shows an example instance of the vehicle routing problem in which there are three routes starting and ending from and to the depot 0.

VRP can be defined formally as a complete directed graph $G = (V, A)$ where $V = \{0, 1, \ldots, n\}$ is the set of vertices and $A$ the set of arcs. The vertex 0 is the depot servicing an identical fleet of vehicles. A non-negative travel cost $c_{ij}$ is associated with every arc $(i, j) \in A, i \neq j$. The problem is symmetrical, so that $c_{ij} = c_{ji}$ for every arc $(i, j)$.

The cost matrix $c$ satisfies the triangle inequality:

$$c_{ik} + c_{kj} \geq c_{ij} \text{ for all } i, j, k \in V$$

so that deviating from the direct $c_{ij}$ link between two vertices $i$ and $j$ is never beneficial. Given these descriptions, we can now define the problem as follows. The problem
objective is to find a minimum $K$ cycles, the cost thereof defined by the sum of their associated arcs, so that

- each vertex apart from the depot is visited exactly once by exactly one vehicle and
- all routes start and end at the depot.

However, the basic VRP model is rather unrealistic: in real-life situations, vehicles cannot carry an infinite amount of goods nor can they travel indefinitely. Thus to realistically apply VRP to real-life situations we need to impose certain restrictions. The most fundamental restriction is that of capacity, which is introduced in the **capacitated vehicle routing problem** (CVRP).

CVRP imposes a maximum capacity for each vehicle. In CVRP, the demands of the customers are known in advance and may not be split. Each vehicle has a capacity $Q$ that collects a demand $d_i$ at each vertex (customer) $i \in V \setminus \{0\}$. The maximum capacity for a route is then defined as follows. Given a route defined by $\langle i_0, i_1, \ldots, i_n \rangle$ be the sequence of the customers in that route and let $i_0$ and $i_n$ represent the depot. Each route satisfies the capacity constraint if $\sum_{j=1}^{n-1} d_{ij} \leq Q$.

Using these constraints, the problem can be formulated into the following mixed integer programming model. In the model, the fleet set $k \in K$ is the fleet of vehicles. The decision variable $x_{ij}^k = 1$ if the arc $(i, j)$ belongs to the route operated by the vehicle $k$, 0 otherwise; $y_i^k$ is the sequence number of node $i$ in the route of vehicle $k$.

$$\text{min} \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}^k$$ (2)

subject to

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1, \quad \forall i \in V \setminus \{0\},$$ (3)

$$\sum_{i \in V \setminus \{0\}} x_{i0}^k = 1, \quad \forall k \in K,$$ (4)

$$\sum_{j \in V \setminus \{0\}} x_{0j}^k = 1, \quad \forall k \in K,$$ (5)

$$\sum_{i \in V} x_{ij}^k - \sum_{i \in V} x_{ji}^k = 0, \quad \forall i \in V \setminus \{0\}, \forall k \in K,$$ (6)

$$\sum_{i \in V} d_i \sum_{k \in K} x_{ij}^k \leq Q, \quad \forall k \in K,$$ (7)
\[ x_{ij}^k (y_{ij}^k + 1 - y_{ij}^k) = 0, \quad \forall i, j \in V \setminus \{0\}, \forall k \in K, \quad (8) \]

\[ y_{0k}^k = 0, \quad \forall k \in K \quad (9) \]

\[ x_{ij}^k \in \{0, 1\}, \quad \forall i, j \in V, \forall k \in K \quad (10) \]

\[ y_{ik}^k \in N, \quad \forall i \in L, \forall k \in K \quad (11) \]

The objective function (2) states that we seek to minimise route length. The constraints (3) state that each vertex is visited exactly once. The constraint sets (4) and (5) impose that each vehicle begins and ends its route at the depot. The constraint set (6) ensures that the same vehicle arrives and departs from each customer it serves. The constraint set (7) is the set of capacity constraints. The constraint set (8) states that the vehicles must travel in a fixed sequence, that is, they cannot return to the previous vertex, and (9) states that the depot must always be the first vertex on any route. Finally, the constraint sets (10) and (11) are the sets of integrity constraints.

CVRP is not the only variant of VRP. In fact, CVRP is but the most fundamental one, many real-life situations are too complex for CVRP. For example, the vehicle routing problem with time windows (VRPTW) adds a time window during which each customer must be serviced. The heterogeneous fleet vehicle routing problem (HFVRP or HVRP) assumes that there are different types of different vehicles, that is, the vehicles are not identical in capacity (or other quality). The vehicle routing problem with pickup and delivery (VRPPDP) turns each customer into depots with the ability to supply goods to vehicles in addition to having a demand. The distance constrained VRP (DVRP or DCVRP) issues a length constraint similar to the capacity constraint, issuing a maximum length for each route. The sheer number and complexity of VRP variants is too large to be explained in detail in this thesis, hence for an overview of different VRP variants we refer the reader to the book by Toth and Vigo [59].

3 Strategies for Vehicle Routing Problems

This section presents an overview of the prevalent strategies for solving VRP. We introduce the reader to existing techniques used in solving VRP by first looking at
exact methods and heuristics in Subsection 3.1. These are called classical approaches, in that they represent a much older methodology, with most of them being developed between 1960 and 1990. We discuss the apparent problems in these approaches and move on to a much newer approach, metaheuristics, in Subsection 3.2. Metaheuristics have seen significant growth in the last two decades [59], and as such are distinct from classical approaches.

3.1 Exact Methods and Classical Heuristics

VRP is a complex problem. As a NP-hard problem, the problem is difficult to solve to optimality. The general strategies for solving VRP are exact methods, heuristics and metaheuristics. Exact methods perform an exhaustive search on all possible solutions attempting to find the best possible solution. Due to the complexity of VRP, the search space, that is, the set of all possible solutions, has a tendency to become excessively large. In turn, this increases the time required for computation. To prevent this, exact methods employ techniques to restrict the search space, effectively reducing the size of the search space. Examples of such techniques are branch-and-cut [37] and branch-and-bound [58]. While exact in the purest sense of the word, exact methods have only provided solutions for up to 100 vehicles [17]. For a more thorough introduction to exact methods in VRP we refer the reader to the review by Toth and Vigo [57].

Given the computational time constraints, in VRP it makes sense to use heuristics: approaches that do not necessarily provide the best possible solution, but provide solutions in an acceptable computational time. In essence, heuristics provide solutions that are good enough: a better solution is likely to exist, but an exhaustive search might be too expensive in terms of time or memory requirements. Heuristics work by approximating solutions that satisfy certain requirements, for example, in VRP, by changing the solution slightly and looking for changes that produce better routes. Heuristics thus perform what could be labeled as educated guesses—there is no guarantee that the discovered solutions are the overall best possible solutions.

In VRP, heuristics can be divided into two distinct categories, construction heuristics and improvement heuristics. Construction heuristics are used to build a route and improvement heuristics are used to improve the constructed solution.

Construction heuristics. Construction heuristics are used to build an initial feasible solution by inserting unvisited (disjoint) vertices into routes with each iteration. Construction heuristics can be further distinguished into single-phase and
two-phase construction heuristics. Single-phase heuristics construct a route in a single step, whereas two-phase heuristics might first split the route into smaller partitions and then build routes for these thereafter. The two examples of single-phase heuristics mentioned here are insertion heuristics and the savings heuristic.

![Fig. 2: Insertion heuristic.](image)

Insertion heuristics were first implemented by Mole and Jameson [35] that sequentially expands the current solution by adding one vertex at a time with each iteration. Figure 2 shows how an insertion heuristic is used to construct the final graph seen in 2e.

![Fig. 3: Savings heuristic.](image)

The savings heuristic by Clarke and Wright [5] starts from a solution where each vehicle is connected directly to the depot as shown in Figure 3. A saving is calculated by looking at the reduction in cost generated merging excess routes. As an example, in Figure 3 for the vertex pair 1 and 2, the travel distance becomes $2c_{1d} + 2c_{2d}$. Merging this route into one becomes $c_{1d} + c_{12} + c_{2d}$. This is then calculated for a savings $s_{12} = 2c_{1d} + 2c_{2d} - (c_{1d} + c_{2d} + c_{12}) = c_{1d} + c_{2d} - c_{12}$. These are calculated for each vertex pair $(i, j)$ and the savings are then sorted in a descending order starting from the largest saving. The routes are then merged iteratively starting from the largest saving until it is no longer possible without breaking the route, that is, no arcs or vertices are disjointed from the graph or the route is no longer feasible. In two-phase construction heuristics, routes are constructed in two phases. The order in which the construction occurs depends on the approach. In route-first, cluster-second heuristics a giant route that visits each vertex is first constructed and is
then clustered into smaller routes. In *cluster-first, route-second* the whole problem is first clustered into smaller clusters and routes then are constructed for each cluster. Examples of two-phase construction heuristics are the sweep algorithm [20] and the Fisher and Jaikumar algorithm [15].

**Improvement heuristics** improve a solution iteratively by creating adjustments to the initial problem. If any improving adjustment is found, the adjustment is implemented. The improving adjustment can be a reduction in tour length or any other beneficial property, e.g., reduction in the number of required vehicles. This is then repeated until we arrive at the local optimum [29]. Examples of such techniques in VRP are k-opt where $k$ arcs are exchanged in each move. Usually $k$ is 2 as in 2-opt or 3 in 3-opt, where two or three arcs are exchanged, respectively. Other widely used search algorithms for VRP and TSP are Lin-Kernighan [32] and Or-opt [46]. For more information on classical heuristics we refer the reader to [7, 29, 59].

### 3.2 Metaheuristics

The above improvement heuristics perform *local searches*. Local searches are heuristics that apply a given heuristic, e.g., 2-opt, to a solution and decide whether to proceed with it or not. A local search looks for improving solutions in a *search neighbourhood*. The concept of local searches and neighbourhoods can be formally defined as follows. Let $S$ be the set of all feasible solutions for a problem $P$. Thus for all solutions $s$ we have $s \in S$. A search neighbourhood $N(s)$ for $s$ is generated by creating solutions that are reachable from $s$ with a move of type $N$ [54].

**Definition 1. Neighbourhood.** A neighbourhood function $N$ is a mapping $N : S \to 2^S$ that assigns to each solution $s$ of $S$ a set of solutions $N(s) \subseteq S$.

The move type $N$ is a slight perturbation of the original problem, and in VRP, it usually involves altering the arcs or vertices of $s$. That is, $N(s)$ defines a set of valid solutions centered around $s$ with radius $\epsilon$ that are all reachable using the move $N$. In a discrete optimisation problem, this gives us the following definition [54]:

**Definition 2.** The neighbourhood $N(s)$ of a solution $s$ is the set $\{ s' | d(s', s) \leq \epsilon \}$ where $d$ is the distance from $s'$ to $s$ given by the move operator.

Thus applying 2-opt to a solution $s$ the neighbourhood $N(s)$ is the set of all 2-opt moves applied to $s$, with $d(s', s)$ being the distance, i.e., the number of applied moves.

---

1. As we will find out later, this quality implies that it is in fact also a metaheuristic, albeit a simple one.
2-opt iterations. Thus generating $N(s)$ using 2-opt once would yield the distance 1, generating it twice would yield 2, and so on.

In a local search, the search neighbourhood $N(s)$ is searched for a better solution $s'$ by looking at all neighbours $s' \in N(S)$. If a better solution $s'$ is found, depending on the heuristic, it can be chosen to replace $s$. If a heuristic picks the first improving solution it encounters, it is called a greedy heuristic. If none of the neighbours in $N(s)$ is better than $s$, we are at a local optimum.

![Fig. 4: Local optima and global optima in a continuous search space for a minimisation problem. The global optimum is characterised by being the overall best local optimum.](image)

**Definition 3. Local optimum.** [54] Relatively to a given neighbouring function $N$, a solution $s \in S$ is a local optimum if it has a better quality than all its neighbours, that is, $f(s) \leq f(s')$ for all $s' \in N(s)$.

In Definition[3], the function $f(s)$ is the objective function, e.g., (2) in VRP. For example in VRP, with the 2-opt heuristic, if we generate a search neighbourhood of size $n$ for $s$ and none of the 2-opt moves improve $s$, $s$ is the local optimum (Figure 4). The whole process of a local search is described in pseudocode in Algorithm[1].

**Algorithm 1 Local search**

**Require:** $s$

```pseudocode
repeat
  Create a search neighbourhood $N(s)$ for $s$
  Select $s' \in N(s)$
  if $s'$ is better than $s$ then
    $s \leftarrow s'$
  end if
until No better solution is found, i.e., at a local optimum
```

\[2\text{For minimisation problems.}\]
Local searches are easy to implement and they can provide solutions quickly, but have a tendency to converge towards local optima. However, it is difficult to estimate the number of iterations a local search will require, as we might encounter a local optimum at the first iteration. To combat this, techniques that try to avoid converging towards local optima have been developed. Of these, the techniques that control local searches to escape local optima are called metaheuristics. Metaheuristics essentially operate a level higher to that of heuristics, in that they control the local searches and have a priori knowledge of the problem—and the heuristic!—enabling them to heuristically guide the heuristic.

It is worth noting that a local search in itself is a metaheuristic—but a very basic one. A local search also has a priori knowledge of the problem, but this knowledge is rather limited. A local search knows how many elements the search neighbourhood \( N(s) \) will contain: the number of possible permutations for each solution for a given heuristic, e.g., 2-opt, can be inferred from the heuristic itself. A local search can also be guided by selecting the worst of the improving solutions, or by accepting the \( n \)th improving solution. In Figure 5 is an illustration of combining a local search with a metaheuristic whereby non-improving moves are accepted with various thresholds, allowing the local search escape from local optima.

![Figure 5: Metaheuristics escaping from local optima.](image)

However, all of this is very elemental. In spite of its metaheuristic capabilities, a local search can do little to avoid local optima. As such, we can classify local searches as the simplest form of metaheuristics. More advanced metaheuristics employ sophisticated techniques to guide and direct local searches in order to avoid or escape local optima. In contrast to local searches that accept only improving moves, metaheuristics avoid converging towards local optima by purposefully accepting

---

3From Greek meta-: “after”, “beyond”, here used in the latter sense.
non-improving solutions (see Figure 5), by changing the search space or by relaxing some of the constraints [54].

Beyond local searches, advanced metaheuristics can be split into two categories: single-solution and population-based metaheuristics [54]. Single-solution metaheuristics improve a single solution iteratively whereas population-based metaheuristics use multiple solutions. Examples of single-solution metaheuristics are simulated annealing [27] that bases its functionality on the annealing process of heated iron and other materials. The principle is that heating a material and then slowly cooling it will result in a stronger crystalline structure. The analogy in SA is that it begins accepting non-improving solutions at a bigger probability, that is, from a heated state, and then gradually lowers the probability of accepting non-improving solutions to escape local optima.

Another efficient single-solution metaheuristic is tabu search [22] that accepts non-improving solutions if none of the moves in the neighbourhood are improving. To prevent repeating this in an infinite cycle, a tabu search maintains a tabu list that contains previously visited moves. Tabu search selects a move and if the move is in the tabu list, the move is discarded. The tabu list size is kept constant to maintain efficiency.

Population-based metaheuristics, on the other hand, use multiple solutions to select improving moves from. Examples of population-based metaheuristics are ant colony optimisation [13] and particle swarm optimisation [26]. Ant colony optimisation mimics the pheromone distribution of ants and particle swarm optimisation uses swarm intelligence observed in the social behaviour of natural organisms, e.g., fish schools and bird flocks in search of food and habitats. A broader category of population-based metaheuristics are evolutionary algorithms that apply artificial evolution to the population, using nature-based processes such as natural selection, mutation and reproduction to evolve the solution iteratively.

In VRP, metaheuristics have produced better results than classical heuristics [4, 33, 47]. The study by Bräysy and Gendreau [3] states that traditional heuristics vary too much in performance and do not provide a technique that is applicable to a sufficient number of problems. The capability of metaheuristics surpasses that of classical heuristics, and the review by Laporte et al. [30] concludes that the future of VRP lies in the field of metaheuristics. For further information on metaheuristics in VRP we refer the reader to [3, 19] and, for metaheuristics in general, to [21, 54].
4 Evolutionary Algorithms

This section explores metaheuristics from an evolutionary perspective. We provide a definition for evolutionary algorithms (EA) and review prominent implementations thereof: evolutionary strategies (ES), genetic algorithms (GA), and memetic algorithms (MA). We begin by providing a background for evolutionary algorithms and exploring their principles in Subsection 4.1, after which we review genetic algorithms and evolutionary strategies in Subsection 4.2 and Subsection 4.3, respectively. In Subsection 4.4, we argue for an alternate but similar approach to GA and then move on to an example implementation, the memetic algorithm, in Subsection 4.5.

4.1 Background and Principles

Evolutionary algorithms draw inspiration from the cycle of evolution observed in nature. Darwin [10] revolutionised modern science with his theory of evolution through natural selection, in which nature and life are seen as a set of life forms trying to compete for survival through adaptation.

This idea of natural selection was applied in optimisation by Fogel [16] as evolutionary programming. Evolutionary programming took a whole decade to gain widespread acceptance and it was not until Holland [24] introduced the genetic algorithm that evolutionary algorithms as a whole became popular.

To implement evolutionary algorithms we assemble solutions into a population. The population consists of individuals which interact and reproduce with one another. This interaction and co-adaptation to their environment is modeled with artificial evolutionary processes, the most fundamental of which is the creation of a new generation. A generation is an iteration of the population, occurring after a preceding one, and it is created by taking individuals from the current population to act as parents and then using nature-based variance, e.g., mutation and gene crossover, to produce new individuals—offspring. The generated offspring are then included into the population. As a result, the population is now larger than originally. To properly implement natural selection, the ultimate step is restoring the population to its original size. This is done by sorting the population using a fitness function, e.g., the objective function. Using this function, we select the most fit individuals to produce offspring. This prunes the weak individuals and restores the population to its original size. Figure 6 is an illustration of the generation process.

To implement an evolutionary algorithm, we must create an initial population.
This can be a random sampling from a given search space, that is, a group of random individuals. After generating the population, the evolutionary process described above is then guided by the following operators:

- selection that selects improving solutions over non-improving ones while maintaining a fixed population size thereby pruning bad solutions,
- reproduction or recombination that mix candidate solutions together to generate new solutions using crossover, and
- mutation that applies random variance to offspring generated by recombination.

In VRP, evolutionary algorithms can be classified into two major disciplines: genetic algorithms and evolutionary strategies. GAs operate on populations of individuals in which information is encoded into genes and chromosomes. These chromosomes are then modified using genetic operators, e.g., mutation and crossover. ES are similar to genetic algorithms, but they tend to rely on mutating the solutions alone. Contrary to a genetic algorithm that mutates the genotype, the set of genes and chromosomes, of a solution. In other words, GAs alter the initial configuration (genes) that is translated to the resulting solution (individual), contrasted with ES that alter the individual.

### 4.2 Genetic Algorithms

As their name implies, GAs use genes to encode information. These genes are assembled into collections or strings that are called chromosomes. The original proposal by Holland [24] uses a string of bits, e.g., 1101 would be a chromosome containing
four genes with the values 1, 1, 0 and 1. These are then subject to mutation and
crossover operators, producing new combinations of their genes. An outline of a
typical GA is given in Algorithm 2.

Algorithm 2 Genetic algorithm

 initialise Generate an initial population of size $N$
 for $n$ generations do
  for $k := 1$ to $N$ do
   select two parent solutions from the population
   recombine these parents into two offspring using crossover
   mutate the offspring with a given probability $p_m$
   include the offspring to a new population
  end for
  evaluate each individual in the new population
  replace the old population with the new one
 end for
 return the best individual

In the first phase an initial solution is created (line 1). This can be a randomised
sampling of solutions. Then, for $n$ times, the process of creating a new generation
takes place $N$ times (lines 2–11). After the reproduction phase (lines 4–7) the popu-
lation is evaluated using the fitness function and the population is restored to size
$N$ by removing the inferior individuals.

4.3 Evolutionary Strategies

Evolutionary strategies [51] resemble genetic algorithms insofar as they adhere to
evolutionary principles and use evolutionary techniques. Gene crossover typical
to GAs is rarely used, and the mutation evolutionary strategies apply are different.
More precisely, ES are characterised by the fact that they model evolution at the level
of phenotypes.
The difference between genotypes and phenotypes is that of configuration and implementation. In nature, genotypes refer to the configuration of all the genes (the genome) of an individual. To draw an analogy from nature, we could consider spruce trees as an example. Norway spruces (of the genus *Picea abies*, pictured right) are coniferous evergreen trees that thrive in mountaineous and moist soil. If we take two Norway spruces sharing the exactly same genotype they are, from a genetic viewpoint, considered identical. If we plant these trees in different environments, one arid and the other moist, the tree planted in the former will grow stunted, and the one in the latter will grow to be a tall, mighty spruce tree. What we have done here is identify their genotypes and their phenotypes. A short tree (or tall) are phenotypes of these genotypes. Thus, the phenotype is the manifestation of an individual’s genes through the modification caused by its environment [12]. While the two spruces share an identical genotype, their phenotypes are distinct.

What do spruce trees have to do with vehicle routing and evolutionary algorithms? The parallels between evolutionary strategies and genetic algorithms are of key importance. Genetic algorithms operate on the individual, i.e., the initial solution with regards to its genes, whereas evolutionary strategies take a phenotypical approach. More specifically, ES treat genes as ordinary values that can be perturbed using real-valued numbers. The way ES do this is they use a decision vector combined with strategy parameters (see Figure 7) where the decision vector $v = (x, \sigma)$ consists of the vector $x$ that is a collection of real-valued variables (the solution)

![Fig. 7: The distinction between the genotype and the phenotype.](image)

![Fig. 8: Norway spruce, *Picea abies*.](image)
and the vector $\sigma$ that consists of strategy parameters. The strategy parameters are a representation of randomly distributed variables, which are used in mutation by substituting $x$ with $x^{t+1} = x^t + N(0, \sigma)$ where $N(0, \sigma)$ is a random Gaussian number with a mean of zero and standard deviation $\sigma$. Mutation is used to provide self-adaptation by mutating both the solution vector $x$ and $\sigma$. Logically, while ES operate on the same set of genes that GA do, the fundamental difference is that ES take a completely orthogonal approach, disregarding the genetic nature of a configuration entirely.

ES have seen competitive applications in VRP. The ES implementation by Mester and Bräysy uses a single individual to generate a single offspring through mutation, a method known as a (1+1) strategy without any recombination. The solutions and strategy parameters are mutated with the use of a guided local search meta-heuristic. The first ES for VRP was developed by Homberger and Gehring. For a better introduction to ES the reader is referred to [1].

4.4 Adding Local Improvement

With traditional GA, there are a few problems to consider. First and foremost, for any given GA, what determines its efficiency and effectiveness are the qualities of its crossover and mutation operators. Tuning these operators is essential, and barring alteration of the initialisation phase, adjusting the crossover and mutation operators is the only way of increasing the performance of GA. An underlying problem in evolutionary algorithms, thus far ignored in this work, is the computational time requirement. The number of possible solutions increases in concert with the scale of the problem. That is, the larger a solution gets, the larger the populations used in GA become. Just as exact algorithms can simply take too long to find an exact result, genetic algorithms can suffer from the same deficit. As a result, while evolutionary algorithms are able to use a wide search space (exploration) due to the diversity found in their populations, their inability to exploit local information only adds to the computational time [45].

These deficits can be overcome by introducing local improvement in the evolutionary process. This would mean selecting individuals to undergo a learning process in which individuals are improved. This would lead to small, but noticeable improvements in the individuals, which in turn speed up the journey towards the optimum, as the amount of required iterations decreases. Before moving on exactly how this is done, we must underline that a GA like this is no longer a pure evolutionary
algorithm. Evolutionary algorithms seek to simulate natural conditions, thus this alteration would remove the “naturalness” of the algorithms.

4.5 Memetic Algorithms

To provide a means for local improvement Moscato [36] proposed the memetic algorithm (MA) by hybridising a GA with a local search. These algorithms are for this reason also called hybrid evolutionary algorithms. The meme was in turn defined by Dawkins [11] as an unit of cultural evolution capable of local refinements. Thus a meme is the set of techniques we use to improve individuals in memetic algorithms.

To implement the concept of cultural evolution we need to slightly alter our genetic algorithm defined in Subsection 4.2 by introducing local improvement, the meme, to the evolutionary cycle. In Algorithm 3 the alteration is replacing the mutation step with a local improvement procedure. In the case of VRP, the local improvement procedure is a local search.

Algorithm 3 Memetic algorithm

| initialise Generate an initial population of size $N$
| for $n$ generations do
| for $k := 1$ to $N$ do
| select two parent solutions from the population randomly using the fitness function
| recombine these parents into two offspring using crossover
| evolve the offspring using a local improvement procedure
| include the offspring to a new population
| end for
| evaluate each individual in the new population
| replace the old population with the new one
| end for
| return the best individual

The local improvement phase can be incorporated in a lot of different ways and for this reason the exact classification of MA has been debated. This thesis omits the classification discourse and concedes to the classification by Krasnogor and Smith [28]: “A memetic algorithm is an evolutionary algorithm that includes one or more local search phases within its evolutionary cycle.” Additionally, as we focus on the
VRP perspective that often uses a local search in place of the mutation step as seen in Algorithm 3, the above definition is well suited for our purposes.

The local improvement is subtle but powerful. MAs have shown to be much more efficient in that they require less computational time and effective in that their results are of a higher quality than traditional evolutionary algorithms. The strength of memetic algorithms lies in their ability to (i) benefit from the exploration abilities of evolutionary algorithms and (ii) exploit local information in a local search to gain better results [28].

The efficiency of MA is evident, but we must point out that traditional—insofar as traditional pertains to originality—memetic algorithms suffer from a certain shortcoming: as the memes employed by these algorithms are tailored to certain needs, memetic algorithms tend to be very ad hoc in their function. That is, the memes themselves are too specific and in VRP this is analogous to using a single local search in a situation where using multiple local searches, each with different qualities and characteristics, would provide a bigger advantage. This would prevent MA in VRP from getting stuck into specific and typical local optima, as using more local searches increases the search space. The evolutionary hyperheuristic developed by Garrido et al. [18] is an example of this technique.

To combat meme specifcness in general MAs, multi-meme implementations employ a variety of different memes each chosen by a hyperheuristic as defined by Cowling et al. [8]. Ong and Keane [44] proposed “meta-Lamarckian” learning that uses Lamarckian learning, a sophisticated evolutionary technique, to adaptively choose the correct memes. Ong et al. [45] defined the adaptive memetic algorithm that defines a process for selecting suitable memes to be applied in learning individuals.

5 Genetic and Memetic Algorithms in VRP

In this section, we apply the previously introduced concepts to VRP. We devise a method for the feasible application of GA to VRP in Subsection 5.1 and study recombination procedures in Subsection 5.2. MAs in VRP are then studied in Subsection 5.4.
5.1 Encoding Routing Problems

Genetic algorithms in VRP owe a great deal to the Travelling Salesman Problem. The latter has seen much more research in the applications of GA. In [38] and [19] we find that TSP-based algorithms can easily be applied on VRP problems. For example, we can partition a VRP instance into subroutes in which there is no depot, rendering the instance into a collection of TSPs. We can now use TSP-based GAs to optimise them and subsequently merge these “sub-TSPs” back into a VRP. How does this work in practice?

The first step is to find a way to represent VRPs in a format that a GA can optimise. While the mathematical model in Section 2 is one way to model VRP instances and the optimisation thereof, a mathematical model does not fit well with the gene-based approach taken by GAs, which operate on chromosomes. For chromosomes, traditional GAs use bit strings to represent solutions. This string is called an encoding. Traditional bit strings are strings of boolean variables, e.g., 0110 is a string consisting of the bits 0, 1, 1 and 0. While a boolean representation of a VRP solution candidate could work, e.g., by using bits to represent the decision variable $x_{ij}^k$ introduced in the mathematical model, [19] argues that the bit string representation is not natural and is better implemented using a path sequence. A path sequence of a VRP solution candidate can be formally defined as a finite integer sequence of its vertices.

**Definition 4. Path representation.** Given a VRP solution candidate, the routes it contains can be modeled as a finite ordered integer sequence $S$, where

$$S := \langle 0, v_1^1, \ldots, v_{n_1}^1, 0, v_1^2, \ldots, v_{n_2}^2, 0, \ldots, v_{n_m}^m, 0 \rangle,$$

where $v_{n_m}^m$ is a vertex in subroute $m$ and $n_m$ is the number of vertices in that subroute. The path begins and ends at the depot 0 and likewise the depot 0 separates routes from each other.

Using this definition, we can now create a path sequence of the VRP instance seen in Figure 1 in Section 2.

**Example.** The depiction seen in Figure 1 can be transformed into the following path sequence. The vertex set $\{1, 2, 3, 4\}$ is grouped into route 1 and the vertex sets $\{5, 6, 7, 8, 9\}$ and $\{10, 11\}$ into routes 2 and 3, respectively. The path sequence of these sets is then

$$\langle 0, 1, 2, 3, 4, 0, 5, 6, 7, 8, 9, 0, 10, 11, 0 \rangle.$$
5.2 Recombination Procedures

Using a path sequence described above, routes can now be modified using mutation and crossover operators. The general distinction between the two is that mutation is about perturbing a single chromosome, similar to a local search, whereas crossover involves exchanging data (genes) between two chromosomes.

**Mutation.** Mutating a path sequence usually involves swapping or shifting vertices around using a fixed probability. An example mutation operator is **remove-and-reinsert** (RAR) that simply relocates the position, i.e., changes $n$ for a vertex $v_n$, and then shifts the remaining vertices accordingly as seen in Figure 9, where 6 is shifted from the position 3 to 5, shifting vertices 7 and 8 left one position.

**Crossover operators.** Crossover operators work by exchanging segments of two chromosomes. In VRP, this is done by exchanging arcs and vertices. The classical one-point crossover determines a cut point at which two chromosomes are split into two parts, and exchanges these parts when creating offspring. However, this might not be feasible in VRP, as it might produce invalid routes. Figure 10 uses the sub-route $\langle 0, 5, 6, 7, 8, 9 \rangle$ and its inverse $\langle 0, 9, 8, 7, 6, 5 \rangle$ and applies one-point crossover on the last two vertices, with the cut point situated after the third vertex. Note that the depot is omitted from the end, as the sequences are seen as cyclical.

As can be seen in Figure 10, both produced routes are invalid. Not only is the first offspring missing vertices, the second one has duplicates! One-point crossover is decidedly too unsophisticated for our needs in VRP and implementing a better crossover operator is necessary. The order crossover (OX) is an order-based operator that was designed to produce valid routes for TSP.
Figure 11 is an example of order crossover applied to the routes found in Figure 10. First, a cut segment (i.e., two cut points creating a subset) is set to include two vertices at the middle. The resulting vertex pair \((6, 7)\) is transferred to the offspring \((11a)\) and from the cut segment onwards, vertices are inserted from parent 2 in the order \(\langle 6, 5, 0, 9, 8 \rangle\). As the vertex 6 is already in the route, it is skipped \((11b)\). When the end is reached, the route wraps cyclically and vertices 9 and 8 are inserted to the beginning of the vertex \((11c)\). The produced route is thus valid and has no duplicates. Many order-based operators crossover akin to the previous one have produced valid offspring routes for TSP \([48]\).

The **Edge-Assembly Crossover** (EAX) is a powerful crossover algorithm that was first implemented for the Travelling Salesman Problem (TSP) by \([42]\) and then modified for CVRP \([38]\). The power stems from the application of condition relaxations. More specifically, EAX loosens the restrictions imposed on the problem formulation, e.g., by allowing can multiple visits to vertices in a route—a violation of the original TSP rules. Consequently, this allows for a more diverse search space, as the number of intermediate solutions becomes larger. This increases the exploratory capabilities of the algorithm.

The actual EAX process consists of a sophisticated two-phase approach. In the first phase, routes are constructed using the edge assembly method\(^4\) under relaxed conditions. Solutions are also refined using a 2-opt local search. In the second phase, the relaxations are removed and construction heuristics are used to construct the routes back such that they adhere to TSP constraints. EAX for TSP can be easily extended to CVRP by neglecting the capacity restrictions. For this implementation we refer the reader to \([38]\).

Examples of other crossover operators are cluster-first, route-second approach in

\[\begin{array}{ccccccc}
\text{Parent 1} & 0 & 5 & 6 & 7 & 8 & 9 \\
\text{Parent 2} & 0 & 9 & 8 & 7 & 6 & 5 \\
\text{Offspring} & 6 & 7 & 5 \\
\end{array}\]

(a) Choosing a cut segment.

\[\begin{array}{ccccccc}
\text{Parent 1} & 0 & 5 & 6 & 7 & 8 & 9 \\
\text{Parent 2} & 0 & 9 & 8 & 7 & 6 & 5 \\
\text{Offspring} & 9 & 8 & 6 & 7 & 5 & 0 \\
\end{array}\]

(b) Insertion from parent 2.

\[\begin{array}{ccccccc}
\text{Parent 1} & 0 & 5 & 6 & 7 & 8 & 9 \\
\text{Parent 2} & 0 & 9 & 8 & 7 & 6 & 5 \\
\text{Offspring} & 6 & 7 & 5 \\
\end{array}\]

(c) Cyclical wrapping.

Fig. 11: Order crossover (OX).

\(^4\)Omitted from this thesis due to scope limitations, see \([42]\) for TSP and \([38]\) for VRP.
GIDEON [56] and GenClust [55]. For a further reading on genetic algorithms in VRP the reader is referred to [2, 48].

5.3 Incorporating Memes

It is of note that as EAX for TSP and CVRP in Subsection 5.2 refines the solutions with a 2-opt heuristic it would indeed mean that the algorithm is in fact a memetic algorithm. Indeed, [39] renames this algorithm to the edge-assembly memetic algorithm, or EAMA, and improves on it by adding more local searches to the local improvement phase. The memetic algorithms studied in [39, 41] make use of the EAX in their local improvement phase.

A local search can be visualised using path sequences with relative ease. As an example, let us visualise 2-opt using a path sequence. Figure 12 below is an illustration of 2-opt operating on the sequence \(\langle 1, 2, 3, 4, 5, 8, 6, 7 \rangle\). In the figure, in (a) and (b) we perform 2-opt by replacing the arcs \((4, 5)\) and \((7, 6)\) with \((4, 6)\) and \((5, 7)\), respectively.

![Figure 12: 2-opt visualisation as a graph and a path sequence.](image)

Given our definition in Subsection 4.5, creating MAs for VRP is a relatively simple procedure. Adding a local improvement step to any GA would increase both its efficiency and effectiveness, and doing so would require but a minor modification to the overall algorithm as can be seen in the differences between Algorithms 2 and 3. While the traditional GA has vast exploratory capabilities, its limited ability to exploit local information provided by local searches is evident. We can thus refine GAs by combining sophisticated crossover algorithms with simple, yet powerful heuristics with little effort. This would suggest that MAs have obsoleted traditional
GAs in VRP.

5.4 Recent Results in Memetic Algorithms

MA have shown to be very effective in VRP and its many variants [31,39,41,50]. This subsection consists of a review of six memetic algorithms for VRP and its variants. Table 1 provides a comparison reference on the key differences between the MA studied in this subsection.

Table 1: A comparison of different MAs in VRP

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Preparatory step</th>
<th>Local searches</th>
<th>LS probability</th>
<th>Crossover</th>
<th>Variant</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAMA</td>
<td>–</td>
<td>2-opt, Swap, Insert</td>
<td>Always</td>
<td>EAX</td>
<td>CVRP</td>
</tr>
<tr>
<td>Penalty EAMA</td>
<td>RMHeuristic¹</td>
<td>2-opt, Relocate²</td>
<td>Always</td>
<td>EAX</td>
<td>VRPTW</td>
</tr>
<tr>
<td>Prins MA</td>
<td>Split</td>
<td>2-opt</td>
<td>pₜₜ</td>
<td>LOX</td>
<td>DVRP</td>
</tr>
<tr>
<td>FMA</td>
<td>Split</td>
<td>2-opt</td>
<td>pₜₜ</td>
<td>OX</td>
<td>HFVRP, VFMP</td>
</tr>
<tr>
<td>MA</td>
<td>PM</td>
<td>Split</td>
<td>2-opt</td>
<td>pₜₜ</td>
<td>OX</td>
</tr>
</tbody>
</table>

¹ Route minimisation heuristic [40]
² Both in-relocate and out-relocate are used

In Table 1, Algorithm signifies the name of the algorithm. The preparatory step indicates the procedure used in initializing the population. Local searches are the local searches used and LS probability represents the probability at which each local search is applied. Lastly, variant is the VRP variant for which the algorithm was designed.

We have chosen the following six MA for the above table: the edge-assembly memetic algorithm (EAMA) for CVRP by Nagata and Bräysy [39], penalty-based EAMA for VRPTW by Nagata et al. [41], hybrid GA by Prins [49] for distance constrained VRP (DCVRP), the FMA and MA|PM for the heterogeneous fleet VRP and vehicle fleet mix problem (VFMP) by Prins [50].

The Edge-assembly Based Memetic Algorithm (EAMA) [39] was derived from EAX for CVRP [38] and the penalty-based EAMA [41] extended this for VRPTW. The latter developed a penalty-based technique for the time window constraint and further split the algorithm into a two-stage process: first a route minimization [40] heuristic is applied to the solution and then EAMA is run. Both algorithms use a local search for every individual. Of the standard 47 benchmarks, the first outperformed 20 existing solutions of these and found the best-known solutions for 24 problems, the second was able to improve 184 best-known solutions (out of 356).

The algorithm by Prins [49] implements a MA for the distance constrained VRP (DVRP). The Prins MA uses a Split procedure to partition the initial VRP into smaller
tours and then applies a 2-opt based local search with a probability $p_m$ in the local improvement phase of the MA. The crossover operator used is linear order crossover (LOX), a derivation of order crossover (OX). This MA outperformed most tabu search heuristics on the 14 Christofides instances and the MA became the best solution for the 20 large-scale DVRP instances by Golden et al. [23].

The FMA and MA|PM algorithms by Prins [50] are two MAs implemented for the heterogeneous fleet vehicle routing problems (HFVRP) and the vehicle fleet mix problem (VFMP). The FMA algorithm uses the Split procedure from the Prins GA [49] and adapts it to the HFVRP. The MA|PM or MA with population management uses a population management algorithm to control the size of the populations. Both algorithms use order crossover [43] and two different local searches based on 2-opt, with fixed probabilities. Of the cases by Taillard [53] and it outperformed most, but not all, algorithms for HFVRP [50].

6 Conclusion and Further Research

In this work, we have reviewed the different evolutionary approaches for VRP. Results show that memetic algorithms are capable of producing excellent result and, in fact, outperform most traditional GAs. Moreover, while the current state of the art algorithms are based on evolution strategies and powerful tabu searches [6], memetic algorithms have shown to be highly competitive in VRP.

The key factors in designing powerful memetic algorithms are (i) the design of the local improvement procedure (ii) the sophistication of the initialization phase and (iii) the possibility of using different local searches adaptively in the improvement phase.

For VRP, there are quite a few possible future paths to consider. From an evolutionary perspective, one of them would be using hyperheuristical and adaptive [45] methods to further improve the memetic algorithms by using altogether different memes and MAs. The evolutionary hyperheuristic for the dynamic VRP by Garudio and Riff [18] has shown to be highly efficient in dynamic VRP. Differential evolution [52] (DE) is a novel evolutionary technique originally designed for continuous optimisation but has been extended to combinatorial optimisation and VRP in [14,34]. DE has shown promising results, and its limited usage in VRP makes DE an interesting candidate for future research. The contrast between hyperheuristics and DE is that DE is a metaheuristic. Moreover, the techniques DE employs in its
evolutionary process are novel and unique, and it while it was designed for continuous optimisation, it can be applied in non-differentiable and discrete optimisation problems as well.

The author would like to thank researchers Tuukka Puranen and Joni Brigatti for their guidance in writing this thesis. This work was supported by TEKES via SCOPE project. This support is accepted with great gratitude.

References


26


