Internal Report 2011–12 December 2011

Universiteit Leiden

Opleiding Informatica

A Meta-Genetic Algorithm for Solving the Capacitated Vehicle Routing Problem

Stefan Wink

Supervisors:
Thomas Bäck
Michael Emmerich

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The Capacitated Vehicle Routing Problem (CVRP) is a widely studied, NP-hard problem with many real-world applications. Exact approaches are infeasible for solving large problem instances, due to the superpolynomial time complexity. Therefore, most solution approaches over the years have been metaheuristics, such as the Genetic Algorithm (GA). This thesis presents a Hybrid GA, which incorporates problem-specific heuristics and domain knowledge into the algorithm. This causes the parameter settings to behave slightly different from regular GAs. To take care of the parameterization, an additional GA is used, acting on the Hybrid GA. This will be referred to as the Meta-GA. In addition to solving all known problems optimally or within 1% of the optimum, it manages to find a new best result within research literature for M-n200-k16, one of the largest problem instances.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The Vehicle Routing Problem (VRP) is a combinatorial optimization problem first described over half a century ago in [Dantzig and Ramser, 1959]. In its most general form the VRP can be described as finding the optimal set of routes for a fleet of vehicles which serve a set of customers. It is a widely studied problem both due to its complexity and thanks to its practical relevance in areas such as transportation, distribution and logistics.

Many solution approaches have been proposed over the years. Exact solutions are only applicable to smaller problem instances due to the NP-hardness [Lenstra and Kan, 1981]. Therefore, most solution approaches have been heuristics or metaheuristics. In this thesis Genetic Algorithms will be used, which belong to the class of Evolutionary Algorithms.

A general overview of the Vehicle Routing Problem will be given in Chapter 2. Genetic Algorithms are discussed in Chapter 3, extending into Chapter 4 on meta techniques. Chapter 5 covers the implementation details and results are presented in Chapter 6. Chapter 7 concludes this thesis.

Chapter 2

Vehicle Routing Problem

The Vehicle Routing Problem is the problem of optimally routing a fleet of vehicles from one or more depots to a set of geographically dispersed customers. A customer has a certain demand and a vehicle has a certain capacity. Each customer has to be assigned to exactly one vehicle in a specific order, without violating the capacity constraints of the vehicles. Figure 2.1 (left) shows a typical problem instance. The green circle is the depot and the black dots represent the customers. Demands and capacities are not depicted here for the sake of simplicity. All vehicles depart from the depot, visit the customers they have been assigned to and return to the depot. The set of routes which forms the minimal total travel cost is referred to as the optimal solution. Figure 2.1 (right) is the optimal solution for the juxtaposed problem instance.

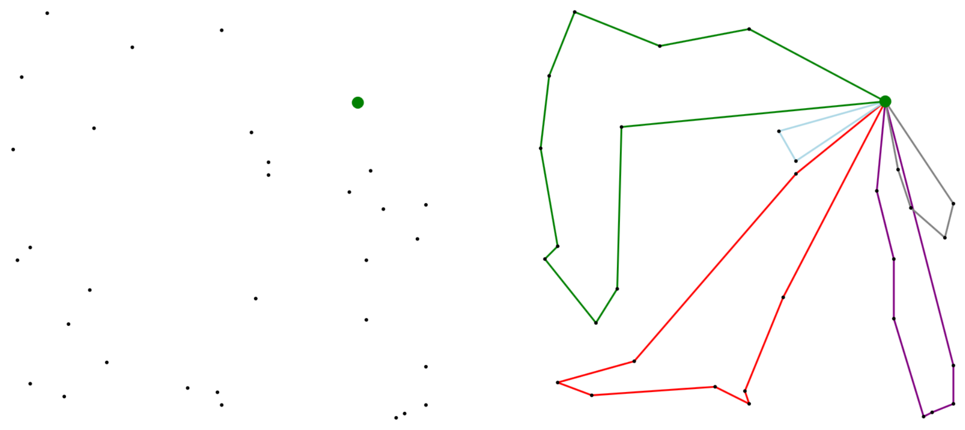


Figure 2.1: VRP problem instance (left) and its optimal solution (right)

The most basic version of VRP is the Capacitated Vehicle Routing Problem (CVRP), which will be used throughout this thesis. In a CVRP there is always one depot and the graph is complete and symmetric, which means that there exists a path from and to every vertex and the distance between vertices is identical both ways. The travel cost is defined as the distance between vertices.

2.1 Problem definition

The mathematical model of the CVRP is defined on a graph $G(V, E)$ where:

- G is complete and symmetric.
- n is the number of customers.
- $V = \{v_0, v_1, \ldots, v_n\}$ is the set of all vertices with depot v_0.
  The set of all customers corresponds to $V - v_0$.
- $E = \{(v_i, v_j) \in V : i \neq j\}$ is the set of all edges.
- $d(v_i) : i \in \{1, ..., n\}$ denotes the demand for customer i.
- $c_{ij} = \delta(v_i, v_j) = \sqrt{|v_{i_x} - v_{j_x}|^2 + |v_{i_y} - v_{j_y}|^2}$ is the travel cost between vertices v_i and v_j.
  $c_{ij} = c_{ji}$ and $c_{ii} = 0$ hold in CVRP.
- $R_i : i \in \{1, ..., m\} = (v_0, v_{i_1}, ..., v_{i_k}, v_0)$ is a route for one vehicle.
  Each route starts and ends at the depot.
- $S = \{R_1, \ldots, R_m\}$ is the set containing all routes, forming the solution.
- $Cost(R_i) = \sum\limits_{j=1}^{i_k - 1}(c_{i_j i_{j+1}}) + c_{0i_1} + c_{i_k 0}$ is the travel cost for R_i.
- Every customer is visited exactly once by one vehicle:
  $\forall i(v_i \in V - v_0 \Rightarrow \exists j : v_i \in R_j \wedge \nexists j \neq k : v_i \in R_j \wedge v_i \in R_k)$
- Total demand in routes does not exceed vehicle capacity K:
  $\forall r(R_r \in S \Rightarrow \sum\limits_{j=1}^{i_k} d(v_{r_j}) \leq K)$

The goal is to minimize the total travel distance, thus the objective function $Cost(S) = \sum\limits_{i=1}^{m} Cost(R_i) = \sum\limits_{i=1}^{m}(\sum\limits_{j=1}^{i_k - 1}(c_{i_j i_{j+1}}) + c_{0i_1} + c_{i_k 0})$ is to be minimized.

2.2 Variants

Many VRP variants exist, usually extending the basic problem by adding constraints in an attempt to make the problem more realistic. The most important ones are described in this section.

- **Multiple Depots**
  Instead of having only one, this variant utilizes multiple depots from which the vehicles can depart. In principle they return to the depot they departed from, but surely there exists an adaptation where this is not a constraint and returning to any depot will suffice.
- **Time Windows**
  Customers must be visited within a certain time window. Each customer is therefore extended with an interval in which it should be served. Failure to visit a customer within that window will result in a penalty or an infeasible route.
- **Pickup and Delivery**
  It is possible for customers to put commodities into the visiting vehicle. This commodity is then taken back to the depot or delivered to another customer en route. Vehicle capacity constraints cannot be violated.
- **Split Delivery**
  The constraint that a customer is visited exactly once by one vehicle is dropped, thus making it possible for a customer to demand more than the truck capacity.

The cost function can be extended with fuel cost, initial running cost per vehicle, labor, fines for not meeting deadlines, etc. The graph does not have to be complete or symmetric either, e.g. in case of one-way roads or roads with an incline.

Nearly all variants can be combined, creating a problem that more closely resembles the routing problem as encountered in real life.

2.3 Related problems

The Traveling Salesman Problem (TSP) is the problem of finding the shortest path for a salesman who wants to visit a number of cities (each city exactly once), departing from and returning to his hometown. CVRP limited to a single vehicle, given enough capacity for the demands of all customers, is equivalent to the TSP.

The goal in the Bin Packing Problem (BPP) is to pack objects of different volumes into a number of bins with a certain capacity that minimizes the number of bins used. This relates to VRP in that customers with different demands should be 'packed' into a route or vehicle which minimizes the amount of vehicles used.

The CVRP can therefore be regarded as a combination of these two problems.

2.4 Complexity

In computational complexity theory, a decision problem is a question in a formal system with a yes-or-no answer, e.g. given a number $n \in \mathbb{N}$, is n prime? A decision problem which can be solved by an algorithm is a *decidable* problem. Decidable decision problems can be categorized by how 'difficult' they are, in terms of solving the problem using the most efficient algorithm. If the problem can be solved by a polynomial-time algorithm (worst case complexity $O(n^k)$), the problem is categorized under complexity class P, which stands for polynomial time. Decidable decision problem for which no polynomial-time algorithm is known but the answer can be checked in polynomial time fall under the complexity class NP (nondeterministic polynomial time). These complexity classes are visualized in Figure 2.2.

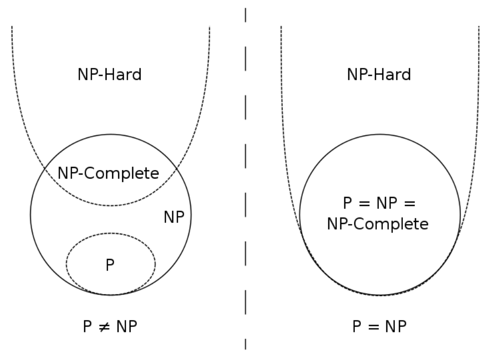


Figure 2.2: Complexity classes

A problem is considered NP-hard if any problem in NP can be *reduced* to it in polynomial time. Reducing a problem to another is translating an instance of a problem to the problem in question. A problem is NP-Complete if it is both NP-hard and in NP. To date, it is unknown whether $P = NP$

and this is a major unsolved problem in computer science. The class of NP-Complete problems is interesting because if any of those problems can be solved in polynomial time, it follows that all problems in NP-Complete are solvable in polynomial time, which would imply $P = NP$. It is generally assumed that problems in NP-Complete are not solvable in polynomial time, partly because people have been trying to do so for the past 50 years. This is known as the NP-hardness assumption [Aaronson, 2006].

The CVRP is NP-hard [Lenstra and Kan, 1981]. The issue in solving such problems is that it takes superpolynomial time to find exact solutions. Currently, exact approaches are feasible for problem sets up to 100 customers. Larger problem sets introduce a phenomenon referred to as 'combinatorial explosion', increasing computational effort very rapidly (i.e. exponentially or worse).

Therefore most attempts on solving CVRP instances involve the use of metaheuristics, which deliver 'good results in acceptable time', even in very large search spaces. This technique is further elaborated in the next chapter.

2.5 Real world applications

Few combinatorial optimization problems have as many practical applications as the VRP. Parcel delivery, transportation of goods, garbage collection, fuel distribution to gas stations, it is everywhere. Efficient routing solutions can save companies both time and money, so the demand for good routing software is very present.

SINTEF, a Norwegian research institute, have done a lot of research on VRP and created a solver which they call Spider, which they have coupled to actual maps, thereby creating a usable solver for real world applications [Hasle, 2010]. It also accepts problem instances used in research literature, which allows for performance comparisons.

An example of a commercially available solver which works on both research problem sets and actual maps is Xtreme Route [Xtreme Route, 2011], which seems to be holding the record for a very large problem instance used in literature.

Chapter 3

Genetic Algorithms

3.1 Background

In 1859 Charles Darwin published 'On the origin of species' in which he introduced the scientific theory that populations of species evolve over the course of generations. Key mechanisms in evolution are natural selection, mutation, genetic drift and gene flow. Darwin stated that individuals with favorable characteristics were more likely to reproduce, thereby supplying the offspring with their favorable genes. In this process the less favorable individuals will disappear from the population over time. This is known as *survival of the fittest*.

Fast forward one hundred years, the 1950s saw the rise of the *metaheuristic* (although the term was first used in [Glover, 1986]), a technique used to find solutions to combinatorial optimization problems by iteratively trying to improve a candidate solution. Metaheuristics can search in very large search spaces and return decent results in reasonable time, though the optimum can not be guaranteed.

Starting in the 1960s, researchers have been proposing metaheuristics inspired by Darwinian evolution. They contain a population of individuals representing candidate solutions, fitness criteria and genetically inspired operators for creating newer generations from the current population. These metaheuristics are referred to as *Evolutionary Algorithms* (which is part of the field *Evolutionary Computation*, which in turn is part of *Natural Computing*). The Genetic Algorithm is one of them.

3.2 The algorithm

First described in [Holland, 1975], the Genetic Algorithm (GA) iteratively improves a population with the evolutionary inspired operators recombination, mutation and selection. The outline is shown in Algorithm 3.1.

Algorithm 3.1 Genetic Algorithm

```
$t \leftarrow 0$
$initialize(P_t)$
$evaluate(P_t)$
**while not** $terminate$ **do**
  $P'_t \leftarrow select{-}mates(P_t)$
  $P''_t \leftarrow recombine(P'_t, p_c)$
  $P'''_t \leftarrow mutate(P''_t, p_m)$
  $evaluate(P'''_t)$
  $P_{t+1} \leftarrow selection(P'''_t)$
  $t \leftarrow t + 1$
**end while**
```

P_t denotes the population at generation t. A population contains a number of individuals representing candidate solutions to a problem. To distinguish good solutions from bad ones, each individual has a *fitness* value, which is calculated during evaluation. The algorithm starts by generating an (often random) initial population and evaluating it. Then the main loop starts: temporary derivative populations are created through recombination and mutation operators and form the next generation. This is repeated until a certain stopping criteria is met, i.e. a pre-set maximum number of generations (loop iterations), a certain fitness value is reached or there is no further improvement in the newly generated population.

Deciding how the fitness value is calculated is a very important step, because the GA will continuously try to improve solutions using the fitness value as its only reference point. The main objective in a problem is usually obvious, so often that is used to calculate the fitness value. It can be extended with additional constraints, assigning a penalty to the fitness value if such constraints are not met.

The number of individuals in a population is a fixed number. Upon initialization, μ individuals are created, forming the 'parents' of that generation. In the recombination phase λ offspring are produced. In the selection phase μ individuals will be chosen from the λ offspring (optionally including the μ parents too) to form the next generation.

Individuals represent candidate solutions. The set of possible solutions in their natural representation is referred to as the *phenotype space*. The evolutionary operators (usually) cannot operate on such phenotypes thus a translation has to be made to *genotypes* using encoding/decoding functions.

3.3 Classical GA

So how exactly do individuals represent a candidate solution? Since the operators in the classical GA operate on binary strings, its genotype should be, according to the classical theory, a binary string as well. Consider a GA trying to find the best input value for the objective function $f(x) = x^2$ ($x \in \{0..15\}$) which is to be maximized. The genotypes should be binary encodings of the phenotypes. The encoding/decoding function here is simply converting integer to binary and vice versa.

3.3.1 Initialization

In this step μ individuals are randomly generated and evaluated:

Genotype	Phenotype	Fitness
0100	4	16
1001	9	81

The genotype is the result of randomly generating a bitstring. The phenotype is the solution the genotype represents (achieved by decoding). Upon evaluation the fitness is calculated, which here is identical to the objective function.

3.3.2 Recombination

Recombination is usually achieved by 1-point crossover. Two parents are selected, a cut point is defined and the portion left of the cut point is copied from the first parent into the offspring, the remainder is copied from the second parent into the offspring:

Parent	01|00
Parent	10|01
Offspring	0101

Parent	1|001
Parent	0|100
Offspring	1100

Parent	010|0
Parent	100|1
Offspring	0101

3.3.3 Mutation

Mutation is done via bit-flip mutation, altering each bit of the genotype with probability p_m, usually defined as $\frac{1}{l}$ such that on average one bit is flipped throughout the genotype:

0101 $\rightarrow$ 01**1**1
1100 $\rightarrow$ 110**1**
0101 $\rightarrow$ 0**0**01

3.3.4 Selection

Prior to this step the offspring created this iteration are evaluated:

Genotype	Phenotype	Fitness
0111	7	49
1101	13	169
0001	1	1

The selection mechanism is called proportional selection (also known as roulette wheel selection). It assigns a probability of being chosen to each individual as $\frac{Fitness}{\Sigma Fitness}$. If both parents and offspring are considered for passing on to the next generation the pool of individuals looks like this:

Genotype	Phenotype	Fitness	$p_{selection}$
0100	4	16	5.1%
1001	9	81	25.6%
0111	7	49	15.5%
1101	13	169	53.5%
0001	1	1	0.3%

Imagine a roulette wheel where each individual is assigned to a part of the wheel. The sizes of the parts are proportional to the selection probability. The population size was μ, so now μ individuals should be picked from the pool. The fictional roulette wheel is spun μ times and the resulting individuals pass on to the next generation.

It's likely that the fittest individuals survive the selection phase so the population for the next generation could look like this:

Genotype	Phenotype	Fitness
1001	9	81
1101	13	169

The process is repeated for new generations. Ultimately the GA should be able to generate the optimal genotype (1111), although this is never guaranteed.

3.4 Characteristics and parameters

3.4.1 Strategy and local optima

In the selection phase individuals are chosen to form the population of the next generation. A distinction can be made in considering only the new offspring or to also include the originating parents. The former is known as *comma strategy*, the latter as *plus strategy*. GAs with μ parents generating λ offspring using comma strategy are denoted (μ,λ)-GAs whereas plus strategy would be a (μ+λ)-GA.

A characteristic of the comma strategy is that it allows for deterioration. If all offspring have worse fitness than their parents the next generation will be worse than the preceding one. While this sounds like a bad idea, it can prevent a population from getting stuck in a *local optimum*.

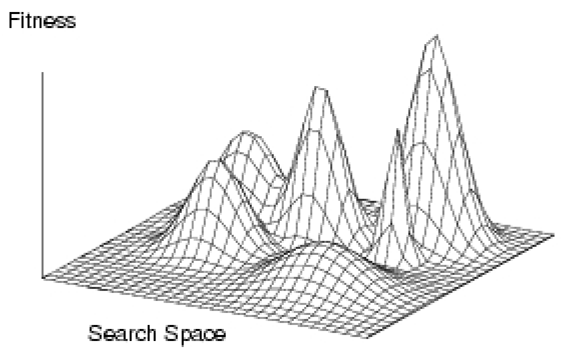


Figure 3.1: Fitness landscape

Figure 3.1 shows a *fitness landscape*, a visualization of the search space, containing several local optima. A (μ+λ)-GA can get stuck in one of those because worse solutions (the ones trying to get out of the local optimum) are eliminated during selection, thereby missing out on the global optimum (depicted as the highest peak).

3.4.2 Population size and selective pressure

Since μ individuals will be selected to form the next generation, λ is usually chosen greater than μ, especially when using comma strategy (else the new population will contain duplicate individuals which kills diversity). If the population size is too small, the GA may not explore enough of the solution space to consistently find good solutions.

Selective pressure is a term used to characterize the emphasis of selection on the best individuals [Bäck, 1994b]. In proportional selection the probability of picking the best individual is equal to its share in the pool (this is inherent to the technique). A widely used selection mechanism that allows for more control in selective pressure is tournament selection. It selects K individuals randomly from the population and then chooses the best individual from that pool. It can be extended by implementing a ranking-based selection within the pool, providing even more control over selective pressure. Both tournament selection and ranking selection are separate selection schemes and they are discussed in [Goldberg and Deb, 1991].

When implementing tournament selection for the algorithm (more details in Chapter 5), ranking selection was inadvertently incorporated. The resulting variation on the tournament selection operator is described in Algorithm 3.2. Choosing $p = 1$ corresponds to the operation of the original tournament selection operator. The algorithm selects one individual from the pool, so within the selection phase of the GA it should be executed multiple times.

Algorithm 3.2 Tournament selection with ranking

```
  choose $K$ (the tournament size) individuals from the population randomly
  choose the best individual from the pool with probability $p$
  choose the second best individual with probability $p * (1 - p)$
  choose the third best individual with probability $p * (1 - p)^2$
  and so on...
```

By adjusting the tournament size the selective pressure can easily be adjusted. Setting $K = 1$ is equivalent to random selection. Increasing the tournament size increases the probability of a good individual will be picked. Population size coupled with the amount of selective pressure determine how fast a GA progresses towards the result.

Figure 3.2 shows the progress of a (20+50)-GA with tournament sizes 2 (left) and 10 (right) on problem instance `A-n32-k5` (more details on problem instances in Chapter 5). Note that the Y-scale is different in both graphs. The optimal value for the problem instance is 784. The green line is the average fitness over the entire population, the red line is the best individual from the population. Neither runs were able to reach the optimal value due

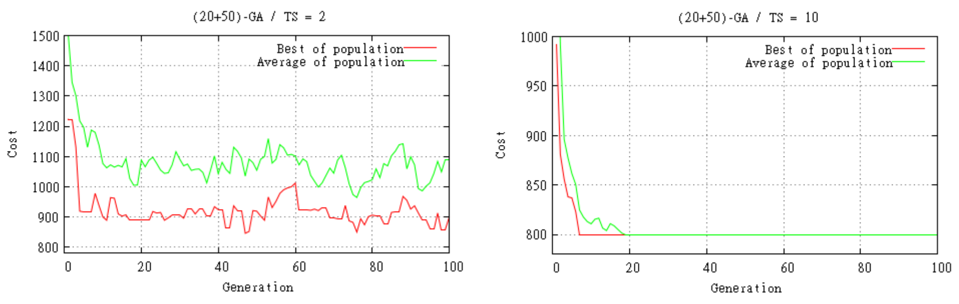


Figure 3.2: No convergence versus premature convergence

to incorrect selective pressure settings. In run depicted by the left plot it was so low that fit individuals were not treated important enough to make it to the next generation. The run shown by the right figure focused too heavily on good individuals, such that within 20 generations the entire population was filled with the same individual.

It should be clear that accompanying population size with a matching tournament size is very important to achieve good results. Unfortunately there is no 'optimal formula' for what the tournament size should be. It should generally be between 2 and 5. Values up to 10 can be applied, but only in large populations. Figure 3.3 shows some runs using different population sizes and tournament sizes. The global best individual is displayed to obtain a cleaner graph.

In the runs represented by the green and yellow lines the selective pressure was too high (premature convergence). The run indicated blue could be improved with a higher pressure. Even though the result is good, larger populations should converge faster than smaller populations (i.e. the red line), since more offspring is generated per generation. Values between 5 and 10 seem to work well for a (100+300)-GA (cyan and purple lines), whereas these values are far too high for a (10+30)-GA.

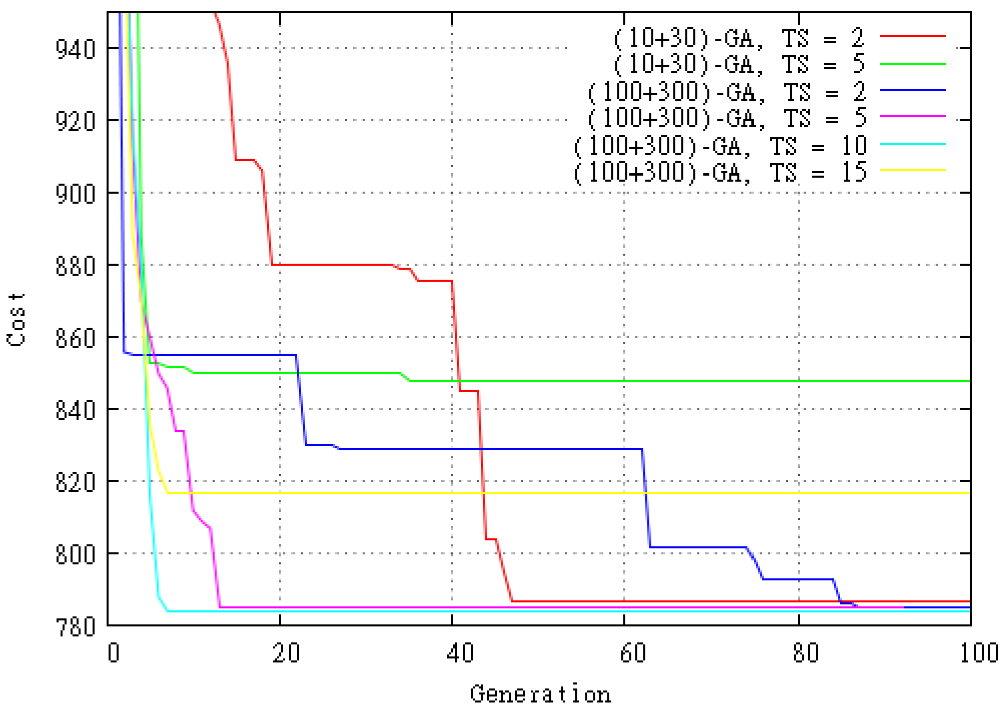


Figure 3.3: Influence of selective pressure

3.5 GA for CVRP

In this section a GA is presented for the CVRP. It will turn out to be different from the Classical GA described in Section 3.3 due to the different individual representations.

3.5.1 Phenotype and genotype

The most natural representation for a CVRP solution is a set of routes with each route containing the customers in the order they are visited. Since each customer is visited only once, concatenating the customers in a solution yields a permutation. Each route should start and end at the depot, but this can be omitted from the representation for simplicity.

Route	Customers
1	2 5
2	3 1 6
3	7 8 4

Encoding permutations to binary form is certainly possible but it does not cope well with the operators of the classical GA. Simple 1-point crossover between two binary encoded permutations is not likely to yield a genotype which is a valid permutation and bit-flip mutation is guaranteed to break the permutation. The GA is not bound to binary operators, however, the genotype should be compatible with the operators. Instead of using repair algorithms, it is better to go with the most natural representation and incorporate the knowledge of the problem into the operators such that they would 'intelligently' avoid building illegal individuals [Michalewicz, 1996].

Two common approaches exist in permutation-based genotype design in VRP. The first approach encodes the phenotype by simply concatenating the customers into one permutation. Decoding is achieved by exhaustively assigning customers to vehicles. The pseudocode for technique is shown in Algorithm 3.3.

The upside of having one permutation is that many operators already exist, of which a good overview can be found in [Bäck et al., 1997]. Also this method takes care of the 'packing' problem, since vehicles are efficiently filled. A major downside is that there are certain configurations for which $x = decode(encode(x))$ does not hold. In other words: the genotype is not able to represent every possible solution. Figure 3.4 illustrates a setting where exhaustive routing is not able to represent the optimal solution (top, travel cost 24). The depot is marked `D` and there are four customers, each

Algorithm 3.3 Exhaustive routing

```
Require: UnroutedCustomers
  $i \leftarrow 1$
  while any UnroutedCustomers do
    if Current customer fits in Route $i$ then
      Append current customer to Route $i$
    else
      $i \leftarrow i + 1$
    end if
  end while
```

with a demand of 1 and the truck capacity is 3. Concatenating (encoding) the optimal solution yields (`1, 2, 3, 4`). Exhaustive routing will then *always* assign the first three customers into the first vehicle and assign the last customer to the next vehicle, resulting in the bottom solution of Figure 3.4 with travel cost 28.

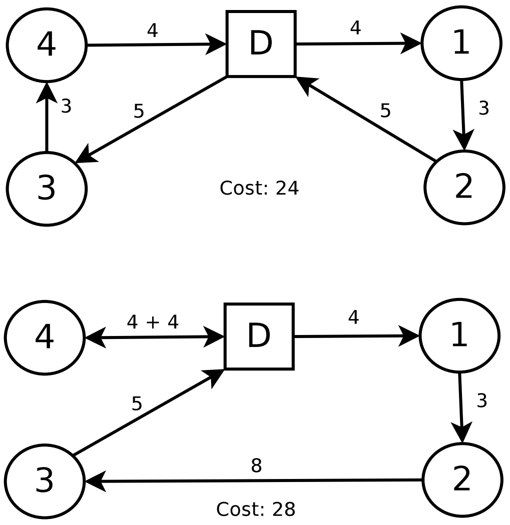


Figure 3.4: The problem with exhaustive routing

Initially this approach was chosen but when it became clear that the genotype could not represent the optimal solution for certain problems, the second approach was adopted: genotype = phenotype. This eliminates the need for encoding/decoding functions, but does introduce the necessity of using specialized operators within the GA. These are elaborated in the next sections.

3.5.2 Fitness value

Each individual has a fitness value assigned to it, measuring the quality of the solution it represents. The objective in CVRP is to minimize the total travel distance, defined in Section 2.1. Problem instances come from TSPLIB, a library for sample instances for TSP and related problems (including CVRP) from various sources. Distances between vertices are Euclidian and rounded to the nearest integer, as described in the TSPLIB documentation [Reinelt, 1995]. The total travel cost for a single route starts at the depot, visits all customers in the specified order and finishes at the depot the route started from. The sum of the travel cost of all routes in the solution yields the fitness value.

Additional constraints can be added to the fitness value, e.g. assigning a penalty in case some routes overlap. Upon inspecting some optimal solutions it turned out that some optimal solutions actually have overlapping routes (Figure 2.1 shows one of them). It was therefore decided not to implement additional constraints in the fitness function.

3.5.3 Useful heuristics

Heuristics are methods that solve often large problems where exhaustive search methods would take too long to compute. They might not always find the best solution, but they are guaranteed to find a good solution in reasonable time. A couple heuristics on graph problems are discussed here. These can be incorporated in the operators of the GA.

3.5.3.1 Push Forward Insertion Heuristic

The Push Forward Insertion Heuristic (PFIH) [Solomon, 1987] is an efficient method to route vehicles across various customers. Being a greedy algorithm, it only looks for the most cost-efficient insertion at each step. This behavior leads PFIH often right into a local optimum. For small sets of customers it works reasonably well, but it cannot be used to solve entire CVRP instances. Nevertheless it is useful when a few customers need to be routed. The outline is given in Algorithm 3.4.

A list of unrouted customers should be supplied. Upon creating a new route the most distant customer is inserted. After that PFIH tries to insert the most cost-efficient feasible (vehicle capacity not exceeded upon insertion) vertex into the route, which can be either a customer or a depot. If returning

Algorithm 3.4 Push Forward Insertion Heuristic

```
$i \leftarrow 1$
while any unrouted customers do
  if Route $i$ is empty then
    Find most distant customer and append it to Route $i$
  else
    $vertex \leftarrow$ most cost-efficient insertion (feasible customers or depot)
    if $vertex$ is a customer then
      Append customer to Route $i$
    else
      $i \leftarrow i + 1$
    end if
  end if
end while
```

to the depot is more cost-efficient then the current vehicle is returned to the depot and a new route is created.

3.5.3.2 2-Opt

2-Opt [Croes, 1958] is a simple local search algorithm which eliminates intersections in routes, originally intended for use in TSP. All pairs of non-adjacent edges are checked for intersections. If they intersect 2-Opt rearranges the edges, creating a route without intersections. Figure 3.5 shows the effect of one iteration: all intersections are removed. The operator can be used to optimize each route, since separate routes are essentially independent TSP problems. Pairs of edges are checked resulting in a complexity of $O(n^2)$.

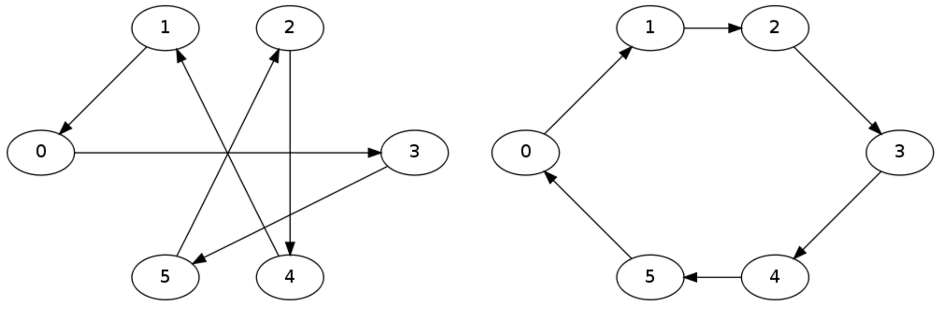


Figure 3.5: Before and after 2-Opt

3.5.4 Initialization

The very first population is created in the initialization step. Each individual is evaluated and assigned a fitness value according to the fitness function. From here on the GA will enter its main iteration cycle. With some knowledge about CVRP, the initial population can be 'steered' into an area in the search space where the optimal solution is expected to be, which can reduce the execution time. However, by generating the initial population entirely random, the GA will be able to scan a larger portion of the search space and is thereby able to 'think outside the box'. It is important that individuals within the initial population differ from each other (have a certain diversity). Initializing using PFIH is a bad idea because that would very likely generate the same individuals over and over again.

3.5.4.1 Random initialization

Creating a random CVRP solution can be easily achieved using exhaustive routing. First, a random permutation of all customers is generated. Then the routes are constructed using exhaustive routing, as described in Algorithm 3.3. The inability of exhaustive routing to represent every possible solution is not really important as creating the initial population is a one-time action. The individuals will be improved through the other operators.

3.5.4.2 Bearing initialization

This operator was created to have an initial population close to a promising area in the search space. Looking at solutions to many VRP problems, a 'butterfly' pattern is visible, illustrated in Figure 3.6.

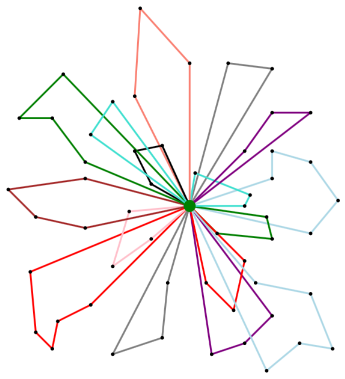


Figure 3.6: Butterfly pattern

This operator attempts to form the individuals of the initial population in the direction of these butterfly shapes. Starting at a certain bearing (i.e. 0°) from the depot node, scan clockwise for customers which will be exhaustively added into the routes, without violating constraints. The idea originates from the scanning operation of a radar. To prevent ending with identical individuals, certain variation should be introduced. This comes from using different start bearings for each individual. Ideally they should be $(360/\mu)°$ apart. The pseudocode is in Algorithm 3.5.

Algorithm 3.5 Bearing initialization

```
Require: $0 \leq StartBearing < 360$
  Calculate bearing for each customer as seen from the depot
  Order the customers by bearing as seen from the depot
  Starting from $StartBearing$, exhaustively create routes
```

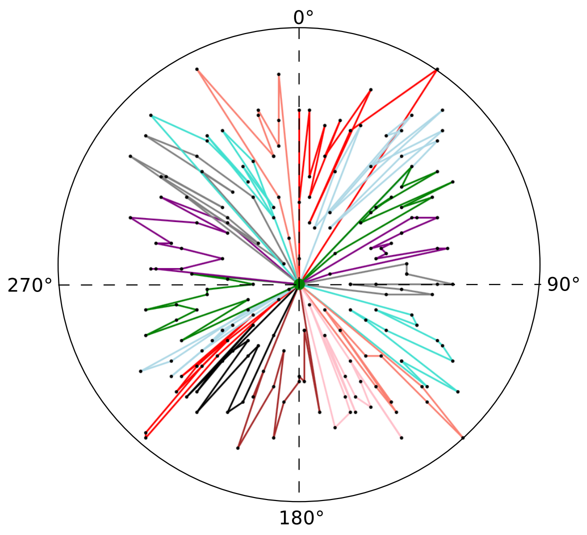


Figure 3.7: An individual created by bearing initialization

Figure 3.7 shows the result of one individual generated using bearing initialization. The depot is in the center, surrounded by the customers. The bearing for each customer is calculated as seen from the depot. Customers in i.e. the top-right quarter of the circle will have bearings varying from 0° to 90°.

A start bearing of 0° was chosen for this instance. The algorithm scans clockwise for customers, starting from the chosen start bearing. Each encountered customer is added to a route using exhaustive routing. Slight variations can be created by using different start bearings.

3.5.5 Recombination

A recombination operator applicable to CVRP should take characteristics from both parent individuals and create a new offspring which forms a valid solution. A number of permutation-based recombination operators have been proposed, but not all are suitable for CVRP. Any permutation can be turned into a CVRP genotype through exhaustive routing, but they often result in solutions that resemble neither parent. An intelligent CVRP recombination operator should recognize existing routes in the parents and use them efficiently, i.e. by inserting non-overlapping routes from both parents into the new offspring. Several of such operators have been proposed in other literature, they will be reviewed in this section.

3.5.5.1 Best Cost Route Crossover

An operator named 'Best Cost Route Crossover' (BCRC) is introduced in [Ombuki et al., 2006]. Designed for VRP with Time Windows, it aims at minimizing the number of vehicles and cost simultaneously while checking feasibility constraints.

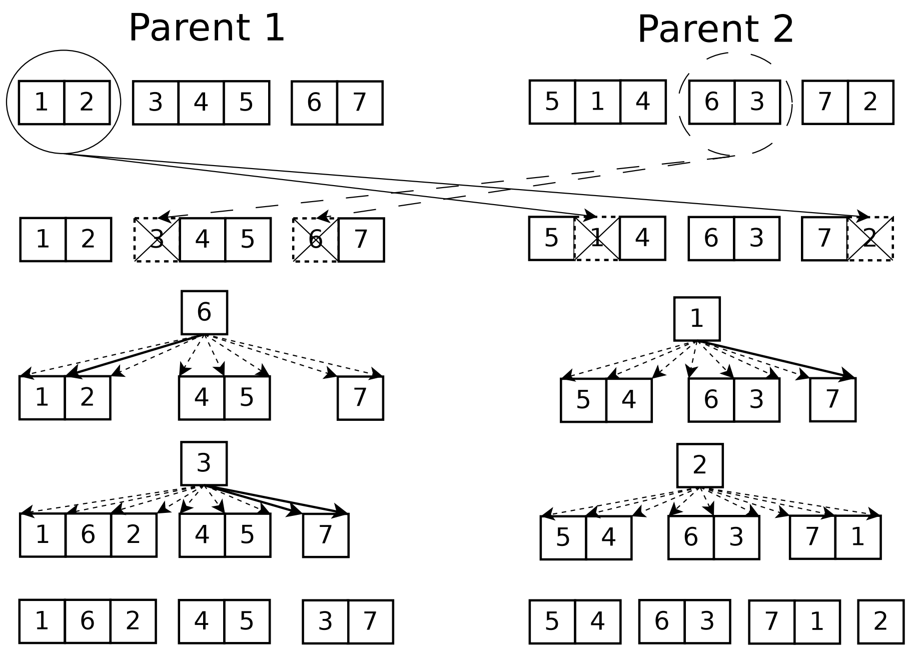


Figure 3.8: Best Cost Route Crossover

Figure 3.8 visualizes how the operator works. Two parents serve as input and two new offspring are generated in the process. One route is picked from each parent of which the elements are then removed from the other parent. In this case, the route from Parent 1 containing customers 1 and 2 is picked, resulting in customers 1 and 2 being removed from Parent 2. The customers which have just been removed will be re-inserted sequentially in the best possible (most cost-efficient) position. Each possible insertion position is evaluated. In case of a tie (equal insertion costs in two or more positions) a random position is chosen within those positions. The operator can create a new route if there are no feasible insertion slots or in case creating a new route results in the minimal increased travel distance.

The offspring resembles the originating parent heavily because it is actually just an optimization of that parent using intelligent insertion techniques. The operator probably works best when re-inserting a small number of customers, because it never looks further than the current customer. When re-inserting a larger number of customers, a technique such as PFIH will likely produce better results.

3.5.5.2 Alvarenga Crossover

Another recombination operator is proposed in [Alvarenga et al., 2007]. It has not been given a name so it will be referred to as 'Alvarenga Crossover'. The operator attempts to take as much entire routes as possible from both parents when generating the new offspring. The outline is given in Algorithm 3.6.

Algorithm 3.6 Alvarenga Crossover

```
while Feasible route exists in both parents do
    Copy random feasible route from Parent 1 into offspring
    Copy random feasible route from Parent 2 into offspring
end while
Attempt to insert unrouted customers in current routes if feasible
Route remaining customers using PFIH
```

In theory this operator creates an offspring resembling both parents equally, although this is very dependent on the problem set. Large problem sets utilizing few vehicles have routes with many customers. If one such route is inserted into the offspring, often no other route is feasible for insertion any more. Routes are often already optimized, especially after a number of iterations, so attempting to insert new customers into existing routes is not likely to happen much.

What is left is a case of 'PFIH to the rescue' to route the remainder, which can be a considerable large group.

Theoretically this operator will rely the least on PFIH in problem sets with many vehicles and small routes. It would then generate offspring with many routes coming directly from both parents. This behavior corresponds to the original intent of the recombination operator.

3.5.6 Mutation

The idea behind the mutation operator is to make a small change to the individual, thereby creating a new offspring which is close to the original in the fitness landscape. In the classical GA this would correspond to flipping a random bit in the genotype bit-string. The genotype for the GA constructed in this section is more complex, which calls for more sophisticated operators. The recombination operators mentioned in last section are aimed towards improving individuals (through evaluating many variations of the current solution), rather than exploring the search space. A simple mutation operator suitable for the current representation is the swapping of customers (within the feasibility constraints), be it intra- or extra-route. This is, however, not likely to improve the solution, it just adds a bit of randomness to it. When using intelligent operators such randomness is undesired, it would only deteriorate the solution created in the recombination step. This section will present two intelligent operators capable of improving solutions even further.

3.5.6.1 Merge routes

This operator breaks up a random number of randomly selected routes and re-orders the now 'route-less' customers using PFIH. Some experiments were done to determine the amount of routes that should be broken up and it turned out that keeping it random yielded the best results. The operator is very efficient in the initial populations because the solutions are often only preliminary. When the solutions evolve the operator is not able to improve in many occasions. Figure 3.9 illustrates the problem.

In preliminary solutions the arrangement of routes is not very efficient as there are many overlapping routes. This improves as the solution evolves. Breaking routes at random and improving the overall solution will only work in a few lucky occasions. If the operator attempts to merge e.g. the top-left and bottom-right routes it will not lead to a better solution, as there is no better way to route them.

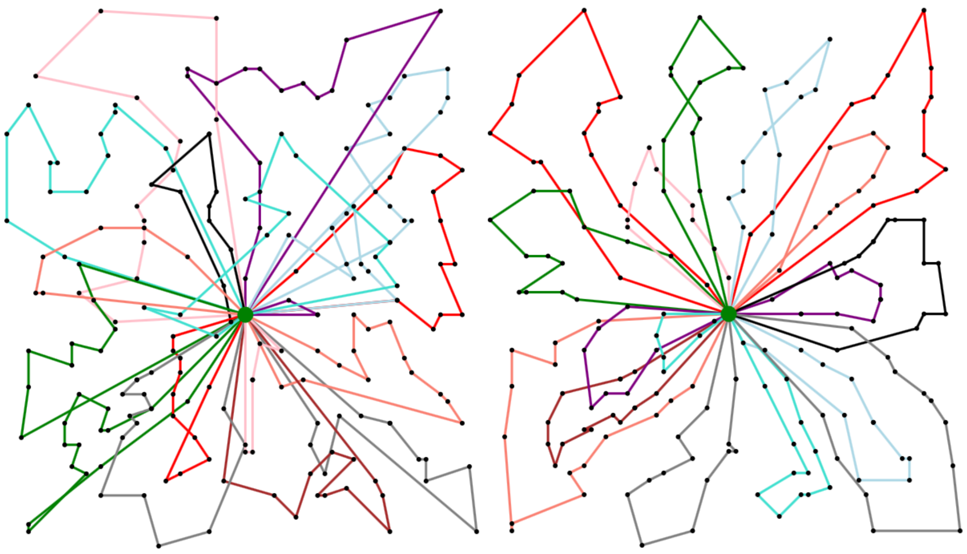


Figure 3.9: A preliminary solution (left) and an evolved solution (right)

3.5.6.2 Adjacent reorder

This operator was implemented in an attempt to address the issue with the ‘Merge routes’ operator. When solutions have evolved the best place to try to improve the solution is in adjacent routes, especially the ones that overlap. The operator works by selecting a random customer. From there the nearest customer in a different route is located. The routes from both customers are then deleted, leaving the customers unrouted. They will be re-routed using PFIH. This way a maximum of two routes will be merged and they are very likely to be adjacent, which is exactly where the solution has a high probability of being improved.

In practice it will probably take longer for the solution to evolve using this operator but the end result might be better. A combination of the two mutation operators would perhaps be best: in the first couple of generations use ‘merge routes’ to quickly improve the initial solutions and once the solutions have evolved use the ‘adjacent reorder’ operator.

3.5.7 Optimization

Besides recombination and mutation an additional step is added to the GA: optimization. In this step each individual will be optimized using the 2-Opt heuristic, described in Section 3.5.3.2, which eliminates intersections within routes. Through the triangle inequality it follows that a route without intersections is always shorter than one with intersections. Thus, optimizing an individual with 2-Opt can only lead to equal or better solutions. The heuristic can be heavy on computation in solutions with large routes due to the $O(n^2)$ complexity for each route.

3.5.8 Selection

The only selection operator used in the GA will be tournament selection with ranking based selection incorporated, described in Section 3.4.2, because of the great controllability over selective pressure. The operator will only be used for selecting new individuals for the next generation. The mates in the recombination step are chosen randomly.

3.5.9 GA classification

The outline for the GA constructed in this section is given in Algorithm 3.7.

Algorithm 3.7 Hybrid GA

```
$t \leftarrow 0$
$initialize(P_t)$
while not $terminate$ do
    $P'_t \leftarrow recombine(P_t, p_c)$
    $P''_t \leftarrow mutate(P'_t, p_m)$
    $P'''_t \leftarrow$ optimize$(P''_t, p_m)$
    $P_{t+1} \leftarrow selection(P'''_t)$
    $t \leftarrow t + 1$
end while
```

The GA uses heuristics in some operators and an additional optimization step is added. GAs incorporating other techniques within their framework (such as heuristics) are called Hybrid GAs, so from now on this GA will be referred to as such.

Chapter 4

Meta-GA

The proposed Hybrid GA in last chapter has many parameters, which need to be set up correctly in order to get good results. This can be very difficult, as the relationship between parameters can be unclear. Instead of manually setting all parameters through trial and error, it is possible to let another GA tune the parameter set. This GA acts on the Hybrid GA and is therefore called a Meta-GA (Greek: $\mu\epsilon\tau\alpha$ = 'after', 'with', 'self'). The concept is visualized in Figure 4.1.

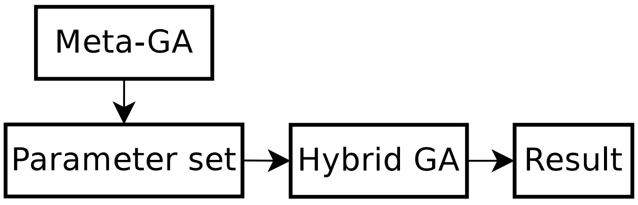


Figure 4.1: Meta-GA model

4.1 Related work

Finding the optimal set of parameters is actually also an issue in the classical GA, where crossover and mutation rate (and to lesser extent population size and tournament size) should be tuned. Unfortunately there is no general optimal setting for these parameters, as different problems call for different representations and different parameter sets [Eiben et al., 2000]. Some 'best practices' in setting these parameters have emerged over the years. Crossover rate is supposedly best kept between 0.6 and 0.95, mutation rate should be somewhere near $1/l$, l being the length of the bit-string [Bäck, 1994a].

One of the first experimentations in tuning the parameter set of a GA using another GA was performed in [Grefenstette, 1986]. This approach has since been dubbed as using a Meta-GA for parameter tuning. Grefenstette's Meta-GA managed to improve the manually set parameters by 3% in a search space of 2^{18} different parameter combinations. The runs were limited to the creation of 2000 meta-individuals (parameter sets) due to the available computing power at that time.

More recent work on parameter tuning using meta-algorithms can be found in [de Landgraaf, 2006], where manual calibration is compared to automated tuning with a Meta-GA. One of the hypotheses, '*using meta-evolutionary algorithms in finding parameters instead of exhaustive search or manual calibration will show a drastic reduction of time required*', was found to be true.

Other than the aforementioned papers, not many people have put much effort in this field. This is a shame, because in 30 years of parameter calibration not much has changed, and questions such as '*are there optimal settings for the parameters of an EA in general?*' and '*are there robust settings for the parameters of an EA that produce good performance over a broad range of fitness landscapes?*' remain without solid answers [De Jong, 2007]. Theoretically, the Meta-GA could give the answers to these questions or even eliminate the need for them altogether, though its effectiveness is sometimes questioned [Clune et al., 2005].

4.2 Meta-GA for the Hybrid GA

This section elaborates on a Meta-GA which targets automatic parameter calibration for the Hybrid GA constructed in Section 3.5. The main characteristic of the Meta-GA is that it should have a small parameter set, because spending extensive time tuning the meta parameter set renders the entire idea pointless. Therefore the Meta-GA will be implemented as a classical GA (described in Section 3.3), using classical operators such as 1-point crossover and bit-flip mutation, both of which require no tuning. Coupled with classical tournament selection, the only parameters for the Meta-GA will be population size, selection strategy and tournament size.

4.2.1 Phenotype and genotype

The following parameters need to be set in order to run the Hybrid GA:

- $\mu \in \mathbb{N}^+$
- Selection strategy $\in$ {Comma, Plus}
- $\lambda \in \mathbb{N}^+$, usually $\mu \ll \lambda$
- Maximum number of generations $\in \mathbb{N}^+$
- Recombination operator $\in$ {BCRC, Alvarenga}
- Mutation operator $\in$ {Merge routes, Adjacent reorder}
- Mutation probability $\in [0, 1]$
- Tournament size $\in \mathbb{N}^+ \leq \lambda$
- Initialization operator $\in$ {Random, Bearing}

Since the Meta-GA will be implemented as a classical GA, the genotype should be a binary string. Thus, all the parameters in the phenotype will have to be encoded. A variable such as μ should vary somewhere in the range $[2 : 1000]$ and λ should be even greater. Using regular binary encoding, this takes up at least 10 bits each, resulting in a very lengthy bit-string. This would create a very large search space for the Meta-GA, since there are 2^l possible configurations, where l is the length of the bit-string. Keeping the bit-string as short as possible results in a smaller search space and faster execution. Not all values in the range really matter (setting μ to 600 or 650 will probably not make a notable difference). By increasing the *step sizes*, a more efficient encoding can be achieved.

In case of μ, the range was adjusted to $[8 : 1024]$ with step size 2^n, resulting in an encoding requiring only 3 bits: $000 \mapsto 8$, $001 \mapsto 16$, $010 \mapsto 32$ up to $111 \mapsto 1024$. In a similar way the amount of bits to represent λ is reduced. Since λ should always be greater than μ, it will be defined as a multiple of μ.

Table 4.1 shows the complete coding table. The length of the resulting bit-string is 15.

Parameter	Range	Bits
μ	[8 : 1024] (step size 2^n)	3
Selection strategy	Comma or Plus selection	1
λ	$[2\mu : 5\mu]$ (step size μ)	2
Recombination operator	BCRC or Alvarenga	1
Mutation operator	Merge routes or Adjacent reorder	1
$p_{mutation}$	[0.3 : 0.9] (step size 0.2)	2
Tournament size	[2 : 17]	4
Initialization operator	Random or Bearing	1
Bit-string		15

Table 4.1: Coding table

4.2.2 Maximum number of generations

The maximum number of generations was omitted from the genotype because it should not be tuned by the Meta-GA for two reasons. First, it would be a shame when the Meta-GA finds the optimal parameter set but only allows it to run for two generations. Second, it should be dependent on the population size. Larger populations reach good results in fewer generations than smaller populations. This can be observed from Figure 3.3 in the previous chapter. This does not necessarily mean that larger populations are faster, since more offspring are generated per generation. So instead of limiting the maximum number of generations, there should be a maximum number of offspring generated. This would give both small and large population sizes equal running time thereby creating a fair comparison.

The amount of individuals generated in a run of the Hybrid GA is $\mu + n * \lambda$ where n is the number of generations. Table 4.2 shows a number of runs for a problem set with 60 customers using a preliminary manually tuned parameter set. The number enclosed in brackets shows roughly how many offspring were generated during execution. There seems to be a trend that larger populations are able to get to the optimum faster than smaller populations, which could be due to the fact that larger populations are able to scan the search space more efficiently (upon initialization it is more distributed among the search space).

The goal is to derive a maximum number of generations based on the number of offspring created. During testing of the Hybrid GA some promising results were achieved on a 200-customer problem after creating around 500,000 individuals. A good result on a large problem instance seems to be a safe upper limit so the maximum number of generations does not have to exceed $500,000/\lambda$.

Hybrid GA	1% from optimum in generation	Optimum in generation
(20+50)	496 (≈25,000)	-
(100+300)	89 (≈27,000)	990 (≈300,000)
(200,500)	25 (≈13,000)	469 (≈235,000)
(500,1000)	26 (≈27,000)	109 (≈110,000)
(1000,3000)	20 (≈61,000)	32 (≈100,000)

Table 4.2: 60 customer problem

4.2.3 Execution time

Table 4.3 shows the time it takes for the Hybrid GA to generate 100,000 individuals for several problem sets with different sizes. Larger problem sets take longer because more computational effort is required in finding a good insertion spot for a customer or evaluating the solution.

Problem instance size	Time to 100,000 individuals
32	1m
44	1m
53	2m
63	2m
78	3m
80	3m
101	8m
151	10m
200	12m
262	13m

Table 4.3: Hybrid GA execution time

The fitness evaluation of one meta-individual corresponds to running the Hybrid GA with the specified parameter set. If the Meta-GA were a (10+30) configuration and the maximum number of generations for the Hybrid GA were set to 100,000/λ, each generation of the Meta-GA would take around an hour for a 44-customer problem to 6 hours for a 200-customer problem. It seems that Grefenstette's problem in falling short of computational resources, now nearly 30 years ago, is still very present.

Thanks to clever coding the search space is not that large ($2^{15} = 32,768$ combinations, fewer than Grefenstette used) so the Meta-GA might not need to run for many generations. Preliminary testing showed that coupled with higher selective pressure good results can be achieved in as little as 10 generations. This puts the execution time in finding the optimal parameter set for an 80-customer problem around 15 hours. Note that these are 'worst case' estimations, where every meta-individual runs for the full generation of

100,000 offspring on the Hybrid GA. In practice, some runs are cut off early because of erroneous selective pressure settings which can lead to premature convergence, such that the total running time of the Meta-GA is shorter.

4.3 Research objective

The main objective of this research is to determine the effectiveness of automated parameter tuning using the Meta-GA approach. This can be distributed among the following points:

- **Performance**: Is the Meta-GA able to match or improve manually tuned parameter sets?
- **Time**: Is the Meta-GA able to match or improve the time it takes to manually tune a parameter set?
- **Robustness**: Do the parameter sets obtained by the Meta-GA perform consistently?

Another interesting question, although only concerning the Hybrid GA, is whether parameter sets that perform well on one problem instance can be re-used on other problem instances, thereby still achieving good results. The previous section shows that, for very large problem instances, the Meta-GA approach simply takes too long. It would be interesting to see how parameter sets obtained from smaller problem instances behave on larger problems.

These questions will be investigated through experiments in Chapter 6.

Chapter 5

Implementation details

This section elaborates on some implementation details of the program that was written to solve the CVRP instances. The GAs described in Chapters 3 and 4 were implemented using C#.

5.1 Problem instances

A large library of (among others) CVRP problem instances is hosted on [Branch and Cut, 2003]. These instances are used throughout literature such that it is easy to compare performance between the different solution approaches. A problem instance contains the coordinates and demands for all nodes (customers and depot). The distances between nodes are in Euclidian distance, but they are rounded to the nearest integer, such that the total travel distance will always be an integer value.

The actual instances originate from papers such as [Augerat et al., 1995] and [Christofides and Eilon, 1969]. Problem sets can be classified into the general 'type' of problem, where the dispersion of customers forms the most important factor, as done in Table 5.1. The difference between a clustered and a dispersed problem instance is visualized in Figure 5.1.

5.2 Visualization

Implementations are often written as console applications and visualization is achieved using gnuplot, but only after the algorithm has finished, making it impossible to see (visually) how the solution evolves. The C# implementation of the program used in the thesis incorporates a graphical user

Set	Instances	Customers	Type
Augerat et al, Set A	27	32-80	Dispersed
Augerat et al, Set B	23	31-78	Clustered
Augerat et al, Set P	24	16-101	Dispersed
Christofides and Eilon, Set E	11	22-101	Dispersed
Christofides et al, Set M	5	101-200	Mixed
Gillet and Johnson	1	252	Dispersed

Table 5.1: Types of problem instances

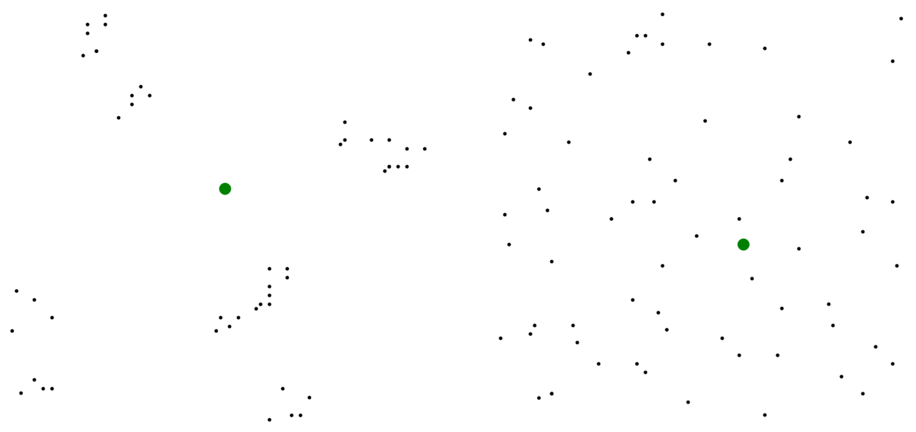
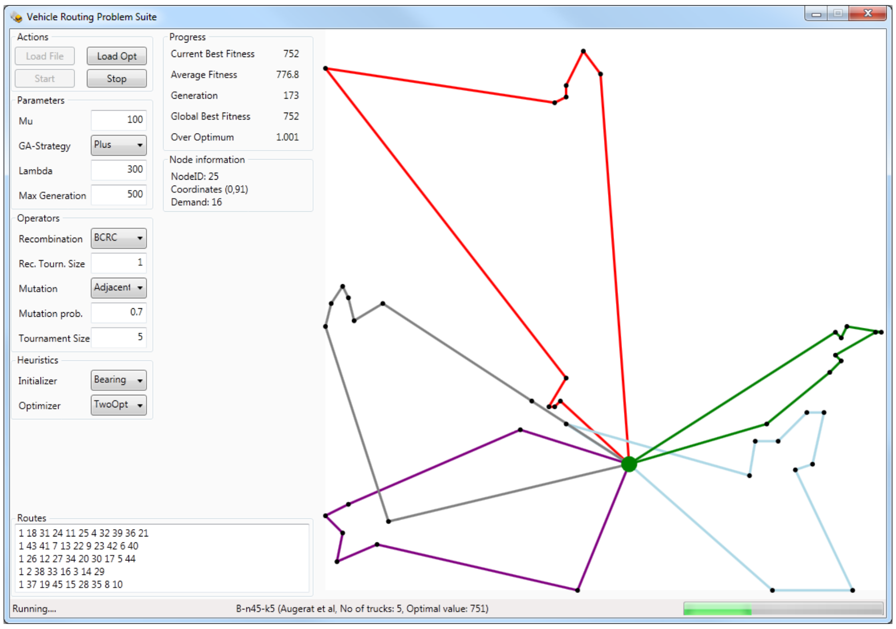


Figure 5.1: Clustered (left) and dispersed (right) problem instances

interface to visualize every intermediate solution. It was initially developed as a 'small bonus', but turned out to be very useful during the development of new mutation operators.

A screenshot of the program which solves instances using the Hybrid GA is shown in Figure 5.2.

All parameters can be set from the user interface. GA progress including current population best fitness, average population fitness, global best fitness, percent over optimum is displayed and hovering over a node shows additional information on the specified node such as coordinates and demand. All visualizations of problem instances and solutions in this thesis are taken as screenshots from this user interface.



Parameters

Mu	100
GA-Strategy	Plus
Lambda	300
Max Generation	500

Operators

Recombination	BCRC
Rec. Tourn. Size	1
Mutation	Adjacent
Mutation prob.	0.7
Tournament Size	5

Heuristics

Initializer	Bearing
Optimizer	TwoOpt

Progress

Current Best Fitness	752
Average Fitness	776.8
Generation	173
Global Best Fitness	752
Over Optimum	1.001

Node information
NodeID: 25
Coordinates (0,91)
Demand: 16

Routes

```
1 18 31 24 11 25 4 32 39 36 21
1 43 41 7 13 22 9 23 42 6 40
1 26 12 27 34 20 30 17 5 44
1 2 38 33 16 3 14 29
1 37 19 45 15 28 35 8 10
```

Running....
B-n45-k5 (Augerat et al, No of trucks: 5, Optimal value: 751)

Figure 5.2: The user interface

5.3 Multi-threading

A generation of the Hybrid GA consists of the following steps: recombination, mutation, optimization, evaluation and selection. Each of these steps involve iterating over λ individuals in the population:

```
// Regular for-loop, serial execution
for (int i = 0; i < lambda; i++) {
    mutate(population.Individuals[i]);
}
```

All iterations within the steps can be executed in parallel, because there is no shared data. It doesn't matter to the GA if the phase is carried out sequentially or in parallel, as long as all individuals get mutated. Microsoft introduced the Task Parallel Library in .NET 4.0 which provide basic forms of structured parallelism through the Parallel class. Basically it is a slightly different syntax for for-loops which allows the content to be executed in parallel, without having to worry about the technical details involved in threading.

```
// C# Parallel extensions
Parallel.For(0, lambda, i => {
    mutate(population.Individuals[i]);
});
```

The same goes for the initialization, recombination, optimization and selection phases, everything can be executed in parallel, creating huge performance gains on multi-core machines. The test machine used has an AMD Phenom II X4 940 quad-core processor operating at 3.0 GHz. The following table shows the improvement when using the Task Parallel Library.

Instance	Serial	Parallel	Speedup
A-n32-k5	26s	10s	2.6x
A-n80-k10	78s	31s	2.6x

Table 5.2: Parallelization speedup in (100+300)-GA to 50 generations

A traditional method for parallelizing GAs is the Master-Slave scheme, where all fitness calculations are done by slave processes. This is only useful when fitness calculation takes very long, i.e. in the meta approach. Another method is the Island model, where every processor runs an independent GA, using a separate subpopulation. Recent work in parallelization includes a hybrid model, reaching speedups of 2.3x [Shinde et al., 2011] on an Intel Core i7 quad-core processor. GPU-accelerated parallelization was implemented in [Zheng et al., 2011], but the final results deteriorated slightly. Both aforementioned papers used C/C++ and they had to implement the parallelization part themselves. The benefit of using a modern language such as C# is that parallelization is included in the library. Utilizing the Task Parallel Library is completely trivial, as can be seen from the code snippets in this section. The gained speedup surpasses that of the mentioned papers and final solutions cannot deteriorate from this parallelization.

Chapter 6

Results

This chapter contains a selection of the results achieved by the Hybrid GA and the Meta-GA, of which the implementation has been discussed in the previous chapters. Experimental runs are performed to give an indication on how altering parameters affects the final result. In the last section the final result table is presented.

6.1 Hybrid GA

This section will focus primarily on experimental runs, with the goal of getting an understanding how the Hybrid GA behaves using different parameter settings.

6.1.1 Population size and selective pressure

Figure 6.1 is a visualization of the performance of the Hybrid GA on problem instance A-n53-k7, a dispersed problem set containing 52 customers. The optimal value is 1010. Different population sizes and varying tournament sizes were used in the runs in order to get an idea of how these parameters influence the overall performance. Best Cost Route Crossover (BCRC) was used for recombination, Adjacent Reorder (AR) was used for mutation and the initial populations were generated randomly.

The fitness values shown are taken as averages from three runs, thus a total of 315 instances of the Hybrid GA were launched to construct the plot. Lower fitness values indicate better solutions. All runs were limited to the generation of 100,000 offspring, giving each run equal execution time. The

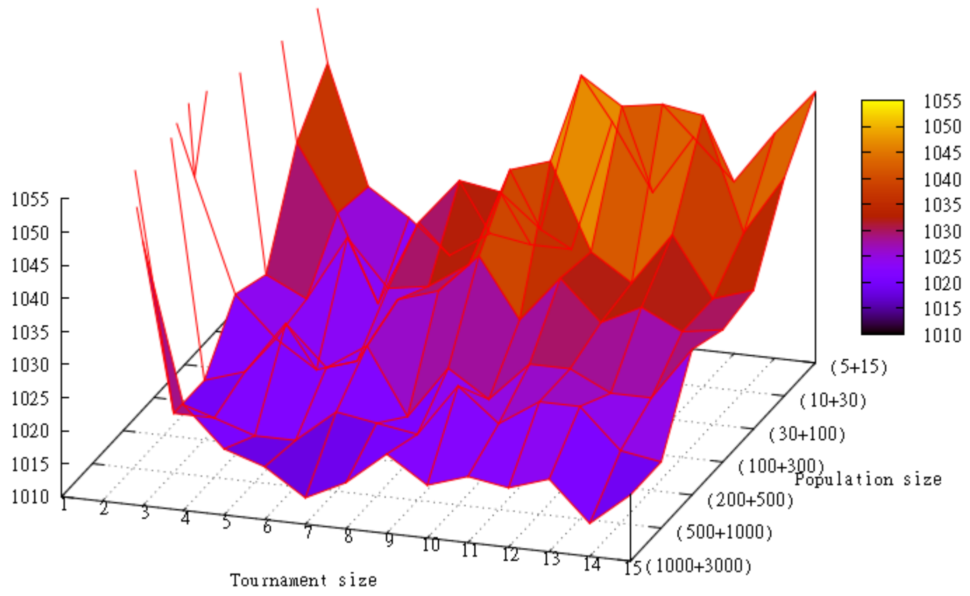


Figure 6.1: Results of various runs on the Hybrid GA using varying population sizes and tournament sizes

plot shows the importance of accompanying population size with a matching tournament size (TS), which was discussed earlier in section 3.4.2. Tournament sizes between 3 and 6 seem to work well when using smaller populations. Increasing the tournament size, thereby increasing the selective pressure, leads to worse results. In very large population sizes, the tournament size can vary a lot without affecting performance, still outperforming smaller population sizes.

Population size	TS	Configuration average
(5+15)	11	1034
(10+30)	5	1020
(30+100)	14	1029
(100+300)	5	1019
(200+500)	6	1023
(200+500)	8	1017
(1000+3000)	7	1014
(1000+3000)	14	1015

Table 6.1: Configurations resulting in the optimum

Table 6.1 shows which configurations resulted in the optimum. The average fitness over three runs from those configurations is also included, to give an impression whether the configuration is able to consistently achieve good results, or if the optimal was just a lucky shot. In this case, smaller population sizes which were able to achieve the optimum generally have a higher average fitness, indicating that the performance of those configurations fluctuates. Configurations with larger population sizes tend to perform more consistently, but it is too early to draw any conclusions from these results, since too many parameters were pre-set.

6.1.2 Recombination and mutation

Two recombination operators have been implemented: Best Cost Route Crossover (BCRC) and Alvarenga Crossover. BCRC focuses heavily on inserting customers in the best possible (cost-wise) location, where Alvarenga Crossover attempts to keep as much of the originating parent individual intact.

These recombination operators can be coupled to the two mutation operators: Merge Routes (MR) and Adjacent Reorder (AR). The Merge Routes operator randomly picks routes and reorders them, Adjacent Reorder focuses on adjacent routes only. It is expected that Merge Routes performs faster, but Adjacent Reorder will eventually find a better solution.

Figure 6.2 shows how the operator combinations perform with a (100+300)-GA on A-n53-k7, which is the same problem instance used in Figure 6.1.

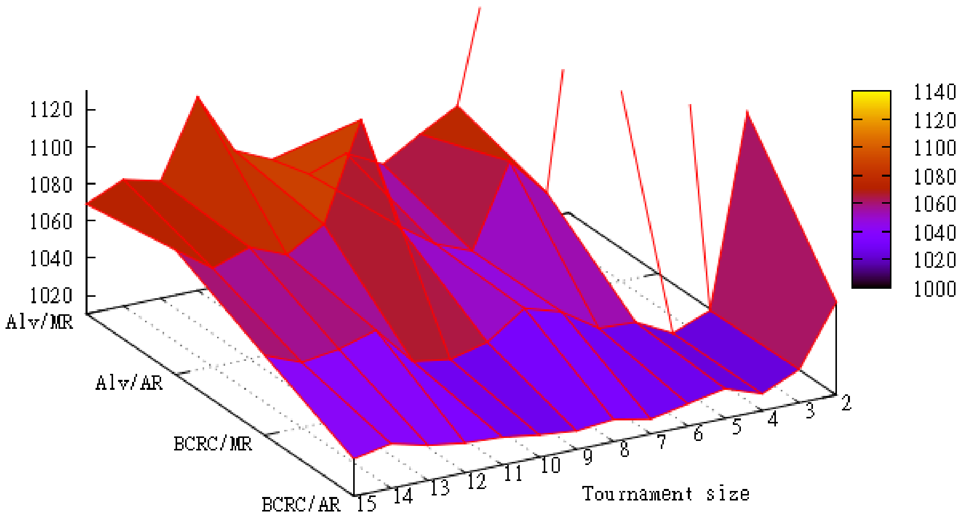


Figure 6.2: Performance of operator combinations with (100+300)-GA on A-n53-k7

From the plot it is very clear that BCRC in combination with Adjacent

Reorder performs best, regardless what tournament size is used. In Figure 6.1 it can be seen that the worst performing BCRC/AR combinations (small population sizes, high selective pressure) yield a fitness of around 1050, which is even better than any Alvarenga Crossover combination manages to achieve using a larger population size. It can be concluded that, at least for this particular problem instance, the combination BCRC/AR consistently results in the best solutions.

6.1.3 Selection strategy

Discussed in Section 3.4.1, the selection strategy determines whether the parents of the current generation are taken into account in the selection phase. Plus strategy includes them, comma strategy omits them. Comma strategy allows for deterioration of the fitness of a population over time, where plus strategy is less likely to do so. This could prevent the comma strategy from getting stuck in local optima.

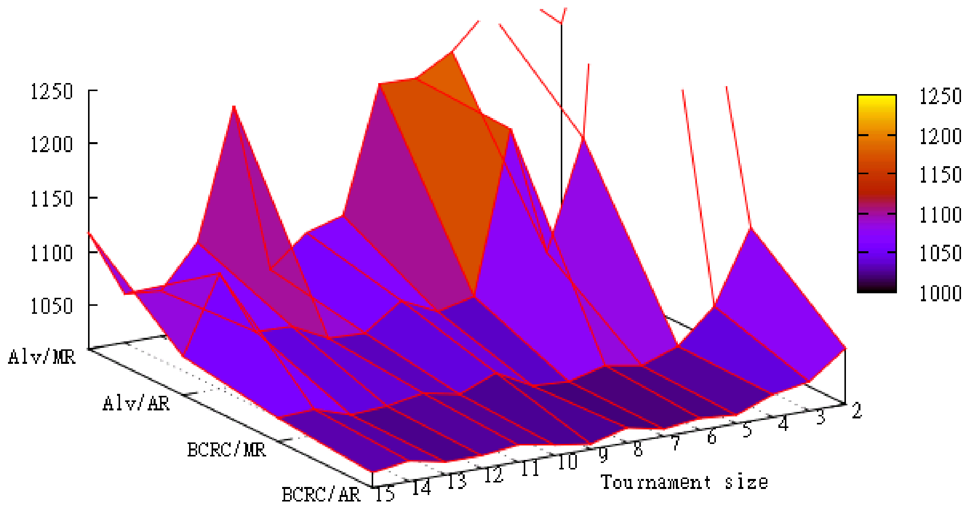


Figure 6.3: Performance of operator combinations with (100,300)-GA on A-n53-k7

Figure 6.3 is the result of essentially the same setup as in Figure 6.2, except it was done with a GA using comma strategy. The shape of the plane is similar, but do note that the Y-scale is different in both plots. The BCRC combinations perform well, but Alvarenga Crossover has some issues with the comma strategy.

Figure 6.4 shows the difference between comma and plus strategy using BCRC/AR (the best performing combination). Overall, comma strategy yields slightly better results.

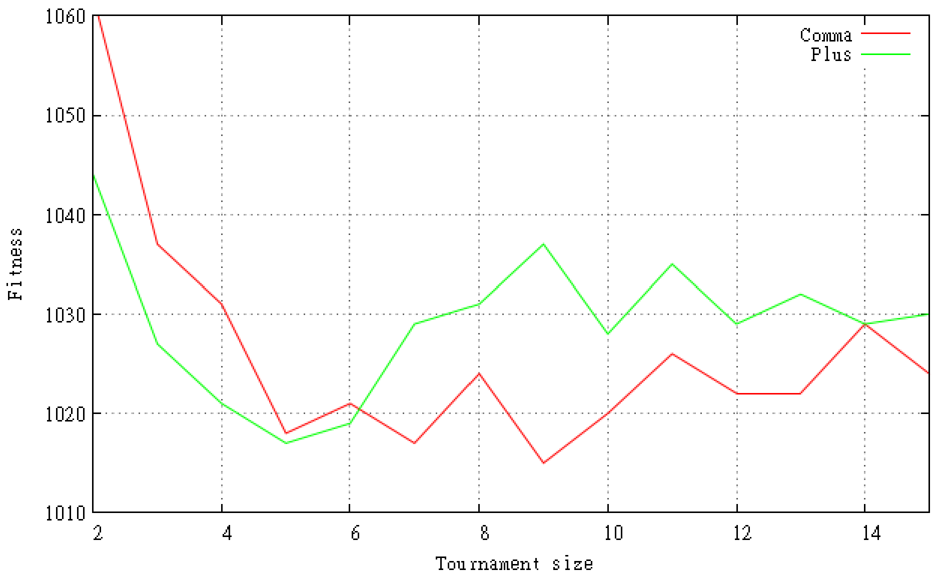


Figure 6.4: Difference between (100+300)-GA and (100,300)-GA using BCRC/AR on A-n53-k7

It is interesting to see how the GA progresses towards the result using the different selection strategies. A typical execution on A-n53-k7 using a tournament size of 5 for both GAs is illustrated in Figure 6.5.

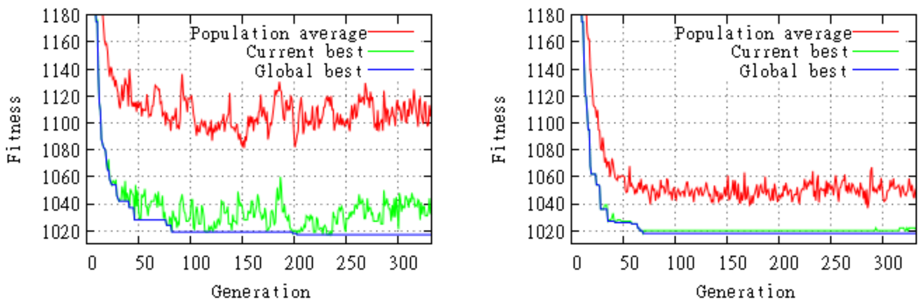


Figure 6.5: Comma (left) and plus (right) selection strategy

The blue line, indicating the global best solution found so far, is almost identical in both runs. The other two lines show major differences. The best individual from the current generation, indicated green, can get worse over time using comma strategy. The fact that the line bounces up and down indicates that this is also happening. Plus selection strategy, on the other hand, continuously tries to select the best individuals in the population, often causing the current generation's best individual to be the global best individual as well. This makes it very prone to getting stuck in a local optima, although the operators could be intelligent enough to prevent this from happening. The GA using comma selection strategy tends to be less 'narrow-minded' in its search for good solutions. From the plot it would appear that it scans over a much larger portion of the search space.

6.1.4 Clustered and dispersed problem instances

Only A-n53-k7, a dispersed problem instance, has been analyzed so far. Figure 6.6 shows the difference in selection strategy B-n57-k9, a clustered problem instance with an optimum of 1598, using BCRC/AR operators and varying tournament sizes. The average of three runs was taken.

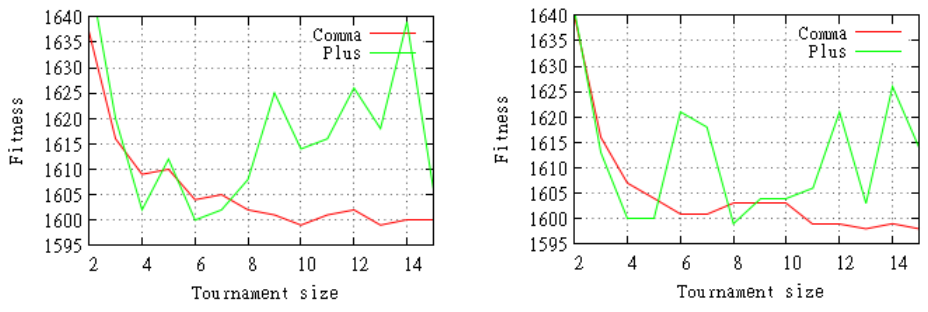


Figure 6.6: $(30 \overset{+}{,} 100)$-GA (left) and $(100 \overset{+}{,} 300)$-GA (right)

The plot is almost identical to Figure 6.4. Again, comma selection strategy has the edge. Its performance is very consistent over the entire range of tournament sizes, where performance decreases in plus selection strategy when using larger tournament sizes. The optimum is found, so it appears there is no modification needed in the parameter settings in order to solve clustered problem instances.

6.1.5 Results

The goal of the previous sections was to gain knowledge on how the different parameter settings affect performance. The performance of the operator combination BCRC/AR was better than the other combinations in the analysis, so that combination will be used throughout this results section. Table 6.2 shows the results of test runs on problem instances in Augerat et al, Set A. In most runs, initially a $(100 \overset{+}{,} 300)$-GA was launched to see how it performed. If the optimum was not reached, the population size was increased, since larger population sizes tend to perform slightly better.

Instance	Optimum	Hybrid GA	Difference	GA	TS
A-n32-k5	784	784	-	(100+300)	5
A-n33-k5	661	661	-	(100+300)	5
A-n33-k6	742	742	-	(1000,3000)	5
A-n34-k5	778	778	-	(50+100)	5
A-n36-k5	799	799	-	(100,300)	5
A-n37-k5	669	669	-	(100,300)	5
A-n37-k6	949	949	-	(100,300)	5
A-n38-k5	730	730	-	(1000,3000)	5
A-n39-k5	822	822	-	(1000+3000)	5
A-n39-k6	831	831	-	(1000,3000)	12
A-n44-k6	937	937	-	(100+300)	5
A-n45-k6	944	948	-	(1000,3000)	12
A-n45-k7	1146	1146	-	(1000,3000)	7
A-n46-k7	914	914	-	(1000,3000)	5
A-n48-k7	1073	1073	-	(1000,3000)	5
A-n53-k7	1010	1010	-	(100+300)	7
A-n54-k7	1167	1168	-	(1000,3000)	15
A-n55-k9	1073	1073	-	(1000,3000)	5
A-n60-k9	1354	1354	-	(100+300)	5
A-n61-k9	1034	1035	+0.1%	(1000,3000)	10
A-n62-k8	1288	1300	+0.9%	(1000,3000)	7
A-n63-k9	1616	1627	+0.7%	(1000,3000)	10
A-n63-k10	1314	1314	-	(1000,3000)	5
A-n64-k9	1401	1411	+0.7%	(1000,3000)	10
A-n65-k9	1174	1178	+0.3%	(1000,3000)	11
A-n69-k9	1159	1159	-	(1000,3000)	15
A-n80-k10	1763	1766	+0.2%	(1000,3000)	10

Table 6.2: Results for instances of Augerat et al, Set A

The overall performance of the Hybrid GA is excellent, as the optimum for every problem instance having fewer than 60 customers is reached and the results of larger instances remain within 1% of the optimum. The execution time varied between seconds for the smaller instances, to up to 15 minutes for larger problem sets. The initialization operator was not significant for the results, so it is not mentioned. The mutation probability varied between 0.6 and 0.9, but never seemed to make an important contribution.

Further test runs were done on Augerat et al, Set B, which contains only clustered problem sets. The parameter settings were kept identical to those of the test run on Set A. Only the largest problem instances were tested. The results are in Table 6.3.

Instance	Optimum	Hybrid GA	Difference	GA	TS
B-n57-k9	1598	1598	-	(100,300)	5
B-n63-k10	1496	1523	+1.8%	(100,300)	10
B-n64-k9	861	861	-	(1000,3000)	10
B-n66-k9	1316	1319	+0.2%	(1000,3000)	10
B-n67-k10	1032	1032	-	(1000,3000)	10
B-n68-k9	1272	1286	+1.1%	(1000,3000)	10
B-n78-k10	1221	1221	-	(1000,3000)	10

Table 6.3: Results for larger instances of Augerat et al, Set B

These results are very good as well. The (1000,3000)-GA with BCRC/AR operators is capable of achieving good results in almost every problem instance, which makes it a very robust parameter set. The only exception lies in B-n63-k10, where smaller population sizes yielded better results. Thus, larger population sizes are not the solution to everything.

6.2 Meta-GA

The previous section contains a lot of analysis on parameter settings, in order to come up with a setting that works well over a range of problem sets. The idea of the meta approach is not having to worry about parameter sets, as they will be figured out automatically by the Meta-GA.

This section covers some test runs and the results achieved by the Meta-GA.

6.2.1 Parameters

The Meta-GA does need some parameters, such as population size and selection strategy. In Chapter 4 it was determined that the Meta-GA cannot have large population sizes due to the excessive computational effort. A (10+30)-GA with maximum of 10 generations coupled with tournament selection (tournament size 5) was chosen as the parameter set for the Meta-GA. Each meta-individual executes an instance of the Hybrid GA. Those instances are limited to the generation of 100,000 individuals, providing equal execution time between small and large population sizes.

6.2.2 Results

Figure 6.7 shows the progress of the Meta-GA on two instances. A tournament size of 5 in a (10+30)-GA makes the selective pressure quite high and because of the plus selection strategy, the current best individual is often identical to the global best individual. High selective pressure causes a GA to find results very rapidly at the risk of premature convergence. In case of the Meta-GA, slow convergence is not desired because computational effort must be limited.

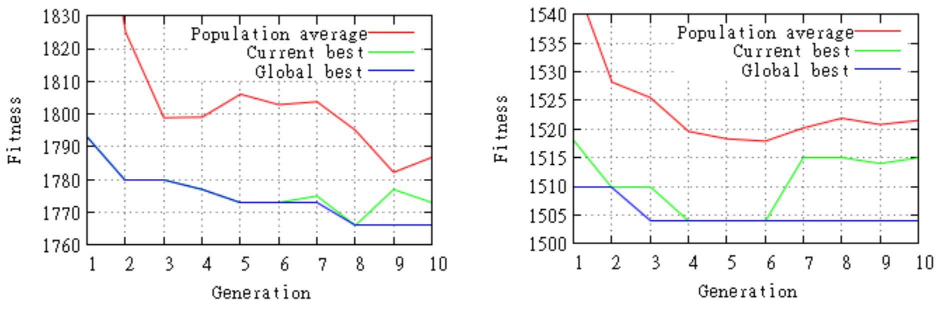


Figure 6.7: Meta-GA progress on A-n80-k10 (left) and B-n63-k10 (right)

The test run depicted in the right plot shows another interesting phenomenon. After the sixth generation the population deteriorates, despite the plus selection strategy. This can happen because a certain parameter set is never guaranteed to return the same result, because it launches an instance of the Hybrid GA. The Meta-GA will then think it has found a good parameter set, which might have been a lucky shot. The algorithm will continue with this seemingly good individual, only to find out later its performance was not consistent. Taking the average over multiple runs for one parameter set is not a solution, because that would multiply the total running time. It is not really a problem, because good results are found either way, perhaps more an anomaly.

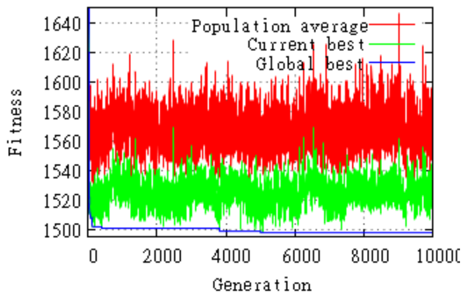
Table 6.4 shows the results of the Meta-GA runs on various problem instances. The results from the manually tuned Hybrid GA are included, they correspond to Tables 6.2 and 6.3. In every occasion, the Meta-GA performs better than or equal to the manually tuned algorithm.

Some parameter sets came as a surprise. For the B sets, containing clustered problem instances, smaller population sizes work better than larger population sizes. Due to the limitation on offspring created, smaller population sizes run for more generations than larger populations, which perhaps gives the edge in clustered instances. After the analysis on the Hybrid GA in the previous section, this came as a bit of a surprise, as a (1000,3000)-GA

Instance	Hybrid GA	Meta-GA	GA	Operators	TS
A-n62-k8	1300	**1292**	(256+256)	BCRC/MR/R	3
A-n63-k9	1627	**1616**	(256+768)	BCRC/AR/R	4
A-n64-k9	1411	1411	(1024,2048)	BCRC/AR/B	14
A-n69-k9	1159	1159	(512,1536)	BCRC/AR/B	8
A-n80-k10	1766	**1765**	(128+640)	BCRC/AR/B	7
B-n63-k10	1523	**1504**	(32,160)	BCRC/AR/R	9
B-n64-k9	861	861	(16,48)	BCRC/AR/R	8
B-n68-k9	1286	**1273**	(16,64)	BCRC/AR/R	7
B-n78-k10	1221	1221	(256,768)	BCRC/AR/B	8

Table 6.4: Meta-GA results

seemed to be able to solve everything. To ensure the parameter set was no lucky shot, a (1000,3000)-GA and a (32,160)-GA were launched on B-n63-k10, without a limitation on the amount of offspring generated. The results are in Figure 6.8.



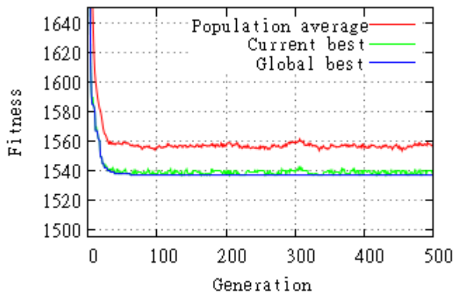


Figure 6.8: Hybrid GA progress on B-n63-k10 with (32,160)-GA (left) and (1000,3000)-GA (right)

The final result of the (32,160)-GA was 1498, which is an improvement over the result mentioned in Table 6.4. The (1000,3000)-GA performed worse than the (100,300)-GA from Table 6.2, as it was only able to reach 1537. It was definitely no glitch, smaller population sizes appear to work better on clustered instances, as the Meta-GA already discovered.

In case of dispersed problem instances, as expected, larger problem instances are the way to go. In terms of operator combinations, BCRC/AR performs best, analogous to the findings in the previous section.

6.3 Final results

The best results from the previous sections have been combined into Table 6.5:

Instance	Optimum	Hybrid/Meta-GA	Difference	GA	TS
A-n60-k9	1354	1354	-	(100+300)	5
A-n61-k9	1034	1035	+0.1%	(1000,3000)	10
A-n62-k8	1288	1292	+0.3%	(256+256)	3
A-n63-k9	1616	1616	-	(256+768)	4
A-n63-k10	1314	1314	-	(1000,3000)	5
A-n64-k9	1401	1411	+0.7%	(1000,3000)	10
A-n65-k9	1174	1178	+0.3%	(1000,3000)	11
A-n69-k9	1159	1159	-	(1000,3000)	15
A-n80-k10	1763	1765	+0.1%	(128+640)	7
B-n57-k9	1598	1598	-	(100,300)	5
B-n63-k10	1496	1504	+0.5%	(32,160)	9
B-n64-k9	861	861	-	(1000,3000)	10
B-n66-k9	1316	1319	+0.2%	(1000,3000)	10
B-n67-k10	1032	1032	-	(1000,3000)	10
B-n68-k9	1272	1273	+0.1%	(16,64)	7
B-n78-k10	1221	1221	-	(1000,3000)	10

Table 6.5: Final results for larger instances of Augerat et al, Set A and B

Even larger problem instances are contained in Christofides et al, Set M. The largest problem instance is probably G-n262-k25, by Gillet and Johnson. Unfortunately, these problem instances are too large to be solved using the Meta-GA, as they require the generation of more than 100,000 individuals to get good results, yielding an execution time of several days.

All problem instances within the M set are dispersed, so the best way to solve them, based on the results from previous sections, is using a (1000,3000)-GA with large tournament sizes and no limitation on the amount of individuals generated. As a result these runs take a couple of hours each. The best results can be found in Table 6.6.

No exact optimum is known for these problem instances because they are simply too large to compute in an exact way. A comparison is made to results which can be found in other research literature. It appears that a new best known is found for M-n200-k16, at least within research literature. Figure 6.9 is a visualization of the solution.

Outside research literature there are commercially available programs, some

Instance	Hybrid GA	Best published	Xtreme Route
M-n151-k12	1025	1021 [Schneider, 2003]	1015
M-n200-k16	**1307**	1335 [Herrero et al., 2010]	1284
M-n200-k17	1303	1296 [TNL, 2011]	1281
G-n262-k25	5628	5574 [Guimarans et al., 2010]	5543

Table 6.6: Results for large instances

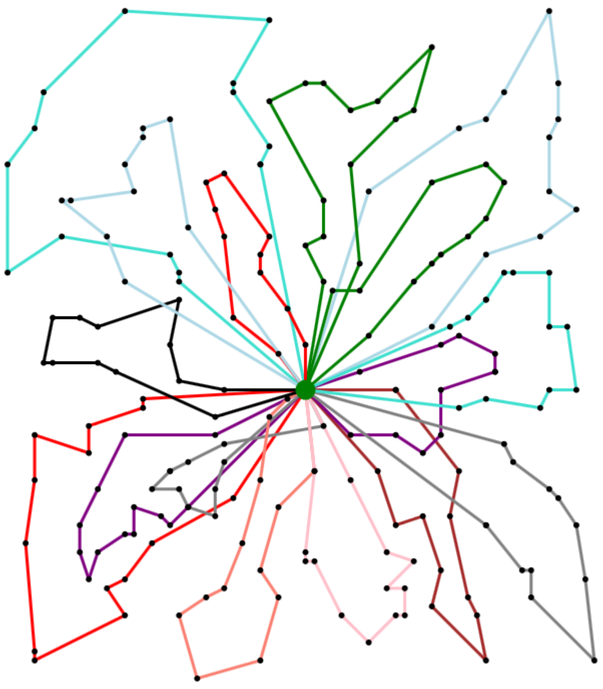


Figure 6.9: Solution to M-n200-k16 with cost 1307

of which also work with the test sets mentioned. One of those is called Xtreme Route [Xtreme Route, 2011]. It is a one-man project which, according to the author, 'uses some local searching' and is 'certainly a branch-and-cut alike algorithm, although there isn't any cutting'. Regardless, it outperforms every approach proposed in research literature.

Chapter 7

Conclusions and Outlook

The initial idea for this thesis was to research how well an order-based GA performs on CVRP. Various new operators have been implemented and analyzed. The overall performance of this 'Hybrid GA' is excellent, nearly all known problems can be solved optimally or within 1% of the known optimum and a new best known within research literature for a large problem instance was found. The drawback was the large amount of operator combinations, so it was decided to extend the research towards a meta-evolutionary approach.

The main focus of this thesis is to determine the effectiveness of automated parameter tuning using the Meta-GA approach. To determine this, three research questions have been defined, which will be answered in this section.

Is the Meta-GA able to match or improve manually tuned parameter sets?
Yes, but only for problem instances up to 100 customers. Larger instances simply take too long to execute (several days), as follows from Section 4.2.3. The Meta-GA results are presented in Table 6.4. All results are better than or equal to the manually tuned parameter sets.

Is the Meta-GA able to match or improve the time it takes to manually tune a parameter set?
Section 6.1 elaborates on the impact of different parameter settings in the Hybrid GA. The plots took several hours to construct, after which some general knowledge is gained on parameter settings for a specific problem instance. It is at that point unknown whether changing parameter settings according to one plot results in the expected behavior on another problem instance. It it assumed that parameter sets behave more or less equally in identical 'types' of problem instances, i.e. clustered and dispersed, but this link only become clear after extensive testing.

A single Meta-GA run takes up to 15 hours and returns the best parameter set and result it can find, requiring no further analysis whatsoever.

If there is no prior knowledge on parameter settings, the answer is a clear 'yes'. Given knowledge, it is possible to find a good parameter set within the time it takes for the Meta-GA to finish. In that case it is a trade-off between time and effort. Running the Meta-GA is effortless but takes time, manually tuning parameters requires effort but less time.

Do the parameter sets obtained by the Meta-GA perform consistently?
Yes, in Figure 6.8, the parameter set found by the Meta-GA was re-used for another run on the Hybrid GA and the result remained good.

So, how effective is the Meta-GA? Given problem instances up to 100 customers and no knowledge about parameter settings, it is extremely useful. But even after analyzing many parameter sets and being under the impression of having a good understanding on the influence of the parameters, the Meta-GA can still surprise with some settings which apparently work very well too, but never came to mind. This is the power of GAs in general: the ability to think 'outside of the box'.

It has to be concluded that the Meta-GA is very useful, but its use is often limited due to time constraints. The approach is very suitable for parallelization and with modern CPU technology focusing heavily on multi-core architecture, the time constraints might be overcome. If time is an issue, the Meta-GA could provide an insight into suitable parameter settings from a single run.

Furthermore, the Meta-GA could be turned into a Mixed-Integer Evolution Strategy (MI-ES), which is able to generate all possible parameter sets. The current Meta-GA approach uses a simplification through the use of step sizes. It has been an assumption that values between those step sizes were not relevant, but this has never been demonstrated.

Bibliography

[Aaronson, 2006] Aaronson, S. (2006). Computational intractability as a law of physics. http://www.scottaaronson.com/talks/anthropic.html.

[Alvarenga et al., 2007] Alvarenga, G. B., Mateus, G. R., and de Tomic, G. (2007). A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows. *Computers & Operations Research*, 34.

[Augerat et al., 1995] Augerat, P., Belenguer, J., Benavent, E., Corberan, A., Naddef, D., and Rinaldi, G. (1995). Computational results with a branch and cut code for the capacitated vehicle routing problem.

[Bäck, 1994a] Bäck, T. (1994a). Parallel optimization of evolutionary algorithms. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*, pages 418–427. Springer-Verlag.

[Bäck, 1994b] Bäck, T. (1994b). Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *In Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 57–62. IEEE Press.

[Bäck et al., 1997] Bäck, T., Fogel, D., and Michalewicz, Z. (1997). *Handbook of evolutionary computation.* Oxford Univ. Press.

[Branch and Cut, 2003] Branch and Cut (2003). Vehicle routing data sets. http://www.coin-or.org/SYMPHONY/branchandcut/VRP/.

[Christofides and Eilon, 1969] Christofides, N. and Eilon, S. (1969). An algorithm for the vehicle dispatching problem.

[Clune et al., 2005] Clune, J., Goings, S., Punch, B., and Goodman, E. (2005). Investigations in meta-GAs: panaceas or pipe dreams? In *GECCO Workshops*, pages 235–241.

[Croes, 1958] Croes, A. (1958). A method for solving traveling salesman problems. *Operations Research*, 5:791–812.

[Dantzig and Ramser, 1959] Dantzig, G. B. and Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, 6(1):80–91.

[De Jong, 2007] De Jong, K. (2007). Parameter setting in eas: a 30 year perspective. In Lobo, F., Lima, C., and Michalewicz, Z., editors, *Parameter Setting in Evolutionary Algorithms*, pages 1–18. Springer.

[de Landgraaf, 2006] de Landgraaf, W. A. (2006). Parameter calibration using meta-algorithms. *Master's thesis.*

[Eiben et al., 2000] Eiben, A. E., Hinterding, R., and Michalewicz, Z. (2000). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3:124–141.

[Glover, 1986] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, 13:533–549.

[Goldberg and Deb, 1991] Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann.

[Grefenstette, 1986] Grefenstette, J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Trans. Syst. Man Cybern.*, 16:122–128.

[Guimarans et al., 2010] Guimarans, D., Herrero, R., Riera, D., Juan, A., and Ramos, J. (2010). Combining probabilistic algorithms, constraint programming and lagrangian relaxation to solve the vehicle routing problem. In *Proceedings of the 17th International RCRA workshop (RCRA 2010): Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion.*

[Hasle, 2010] Hasle, G. (2010). Vehicle routing in practice. Presentation at XVIII EWGLA 2010, Naples, Italy.

[Herrero et al., 2010] Herrero, R., Guimarans, D., Ramos, J. J., and Padrón, S. (2010). A variable neighbourhood search combining constraint programming and lagrangean relaxation for solving routing problems. In *2010 Summer Simulation Multiconference*, SummerSim '10, pages 379–386. Society for Computer Simulation International.

[Holland, 1975] Holland, J. (1975). *Adaptation in natural and artificial systems.* University of Michigan Press.

[Lenstra and Kan, 1981] Lenstra, J. K. and Kan, A. H. G. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227.

[Michalewicz, 1996] Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer.

[Ombuki et al., 2006] Ombuki, B., Ross, B. J., and Hanshar, F. (2006). Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, 24:17–30.

[Reinelt, 1995] Reinelt, G. (1995). TSPLIB95.

[Schneider, 2003] Schneider, J. (2003). Searching for backbones - a high-performance parallel algorithm for solving combinatorial optimization problems. *Future Generation Computer Systems*, 19:121–131.

[Shinde et al., 2011] Shinde, G. N., Jagtap, S. B., and Pani, S. K. (2011). Parallelizing multi-objective evolutionary genetic algorithms. In *Proceedings of the World Congress on Engineering 2011 Vol I.*

[Solomon, 1987] Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35:254–265.

[TNL, 2011] TNL (2011). http://140.113.119.114/. Transportation Network Laboratory, National Chiao Tung University (NCTU), China (Taiwan).

[Xtreme Route, 2011] Xtreme Route (2011). http://www.xtremeroute.com/.

[Zheng et al., 2011] Zheng, L., Lu, Y., Ding, M., Shen, Y., Guo, M., and Guo, S. (2011). Architecture-based performance evaluation of genetic algorithms on multi/many-core systems. In *14th IEEE International Conference on Computational Science and Engineering.*

Appendix A

Solutions

Note that node *IDs*, rather than *indices*, are mentioned. Thus, the depot has ID 1 and the first customer has ID 2.

```
M-n151-k12:
Route #1: 1 53 107 83 49 125 47 126 46 9 115 84 19 90 1
Route #2: 1 133 2 52 104 10 121 82 34 103 51 112 1
Route #3: 1 78 4 130 80 79 35 136 36 137 66 72 67 21 129 123 1
Route #4: 1 29 77 117 69 81 151 122 30 25 135 110 13 139 1
Route #5: 1 106 111 5 140 40 68 26 56 131 55 150 27 1
Route #6: 1 97 105 100 62 17 87 114 18 85 6 119 61 1
Route #7: 1 8 48 37 144 50 65 12 108 20 124 63 149 89 1
Route #8: 1 1 147 128 32 11 109 127 64 91 33 132 31 71 102 70 28 1
Route #9: 1 41 22 74 73 75 76 57 24 134 23 42 146 116 3 138 1
Route #10: 1 88 145 58 16 44 43 143 15 141 39 120 45 142 92 101 118 1
Route #11: 1 148 7 60 94 86 99 38 93 98 96 95 1
Route #12: 1 54 59 14 113 1
Cost 1025

M-n200-k16:
Route #1: 1 61 119 85 174 18 114 87 141 39 15 120 193 101 118 1
Route #2: 1 123 104 162 72 66 137 36 136 165 35 79 186 77 1
Route #3: 1 70 163 102 71 91 33 132 161 129 67 189 21 31 2 1
Route #4: 1 38 99 86 194 92 192 45 142 17 62 6 7 1
Route #5: 1 95 96 98 93 152 60 94 105 100 97 184 54 1
Route #6: 1 107 195 8 125 47 37 144 50 65 182 64 127 109 190 11 1
Route #7: 1 153 138 145 58 16 44 143 43 173 88 14 113 157 1
Route #8: 1 41 74 73 172 75 134 23 42 146 116 179 3 59 1
Route #9: 1 181 22 199 198 76 24 187 57 111 180 139 155 1
Route #10: 1 148 84 200 126 46 175 9 115 154 19 167 90 1
Route #11: 1 28 168 128 191 32 160 63 149 89 53 147 1
Route #12: 1 83 49 48 169 124 20 108 176 12 183 1
Route #13: 1 112 51 103 158 34 82 121 10 52 177 133 1
Route #14: 1 106 27 150 196 13 151 81 69 117 185 29 1
Route #15: 1 5 156 140 188 40 68 171 26 56 166 131 55 1
Route #16: 1 197 78 159 4 80 130 170 122 30 25 164 135 178 110 1
```

```
Cost 1307

M-n200-k17:
Route #1: 1 19 115 9 175 47 125 48 169 49 83 154 53 147 1
Route #2: 1 107 195 8 124 20 37 144 50 65 12 176 108 183 1
Route #3: 1 112 158 34 82 121 10 162 104 52 2 133 1
Route #4: 1 99 194 92 192 142 45 141 39 15 120 193 101 38 152 93 118 1
Route #5: 1 29 185 197 117 78 4 159 103 51 177 28 1
Route #6: 1 70 123 129 21 189 67 72 66 137 36 136 165 35 79 1
Route #7: 1 54 106 196 110 178 151 81 69 13 139 155 1
Route #8: 1 77 186 80 130 170 122 30 25 164 135 131 55 150 27 1
Route #9: 1 180 166 56 26 171 68 40 188 140 156 5 111 199 1
Route #10: 1 113 1
Route #11: 1 7 97 60 94 86 62 174 85 6 119 1
Route #12: 1 90 167 61 84 200 126 46 18 114 87 17 100 105 1
Route #13: 1 157 148 184 95 96 98 88 153 59 1
Route #14: 1 181 22 73 198 57 187 24 76 134 23 75 41 1
Route #15: 1 14 138 3 116 179 145 173 43 143 44 16 58 146 42 172 74 1
Route #16: 1 168 109 91 127 64 182 33 132 161 31 71 102 163 1
Route #17: 1 89 149 63 160 190 11 32 191 128 1
Cost 1303

G-n262-k25:
Route #1: 1 167 235 69 143 233 232 203 140 2 230 72 32 206 134 6 205 1
Route #2: 1 141 165 257 189 35 96 183 29 161 26 142 1
Route #3: 1 40 234 33 79 148 160 135 159 61 111 30 98 76 1
Route #4: 1 24 49 144 15 108 213 46 188 129 121 1
Route #5: 1 176 259 25 246 128 41 18 56 48 57 243 1
Route #6: 1 112 237 197 175 99 241 118 254 258 223 166 1
Route #7: 1 19 13 212 91 62 222 130 131 50 125 256 174 261 28 1
Route #8: 1 208 73 186 20 80 262 192 31 1
Route #9: 1 34 253 245 116 146 101 139 22 97 1
Route #10: 1 210 100 71 109 156 77 155 88 5 82 187 173 1
Route #11: 1 153 145 193 185 219 120 240 36 260 157 8 75 1
Route #12: 1 191 9 4 216 105 214 113 1
Route #13: 1 127 251 47 92 51 95 152 150 87 248 89 78 94 201 200 117 1
Route #14: 1 247 209 181 64 217 162 180 23 45 119 124
Route #15: 1 123 227 202 52 68 133 106 86 38 7
Route #16: 1 147 103 16 220 137 177 39 224 44 1
Route #17: 1 65 194 102 14 168 90 249 122 107 190 154 149 252 1
Route #18: 1 3 171 231 10 27 115 126 1
Route #19: 1 74 199 55 138 229 54 66 207 84 196 1
Route #20: 1 59 184 151 221 37 218 1
Route #21: 1 17 93 179 211 42 110 21 228 85 1
Route #22: 1 136 53 163 60 172 182 170 244 1
Route #23: 1 70 178 238 63 250 158 225 164 1
Route #24: 1 255 67 169 104 114 81 195 198 43 11 239 12 242 236 1
Route #25: 1 204 58 132 215 226 83 1
Cost 5628
```