Universität Augsburg
Fakultät für Angewandte Informatik

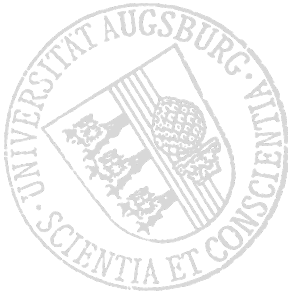
Bachelorarbeit

für die Prüfung zum Bachelor of Science
im Studiengang Ingenieurinformatik

Improvement and Analysis of a Genetic Algorithm for the Virtual Network Embedding Problem

Concept and Implementation

Raphael von Lottner



Software Methodologies
for Distributed Systems

Raphael von Lottner

Improvement and Analysis of a Genetic Algorithm for the Virtual Network Embedding Problem

: **Prof. Dr. Bernhard Bauer**,
Fakultät für Angewandte Informatik, Universität Augsburg

: **Prof. Dr. Lorenz**,
Fakultät für Angewandte Informatik, Universität Augsburg

: **Andrea Fendt**,
Fakultät für Angewandte Informatik, Universität Augsburg

: 1436707

: 31. März 2020

Abstract

Abstract

The Internet is ossifying - a bottleneck for future technologies and services. The needed flexibility and the freedom to experiment with promising new architectures, however, could be achieved by network virtualization. Therein the allocation of resources between physical and virtual networks plays a key role. The profitability of realizing virtual networks depends on the optimization of this allocation, which turns out to be a complex combinatorial problem with today's widely varying network requirements. Genetic Algorithms (GAs) have many desirable characteristics to tackle this problem. Their meta-heuristic approach, suitability for distributed and parallel implementations, as well as their successful application in a wide range of problems make them a promising candidate. This work will investigate alternative genetic mechanisms by evaluating an alternative crossover operator and two different mutation operators. Furthermore, it will consider a different generation replacement strategy that puts greater selective pressure on its population and a more competitive selection mechanism. Finally, this work will consider an additional termination criteria and use two test environments based on a model that focuses on CPU and bandwidth to evaluate these concepts. The author comes to the conclusion that many of the different genetic operators explored can be useful, with the Tournament Selection operator being the most promising candidate identified during these investigations.

Contents

1 Introduction

Future and current technologies are highly influenced by the capabilities of their communication network solutions and as such, with the Internet [45, p.1]. The continuous development of the Internet depends on innovation-friendly flexibility, in which network virtualization plays a major role [24, p.1]. This thesis gives an overview of the central challenge of Network Virtualization (NV) and investigates an evolutionary approach as a suitable candidate solution.

1.1 Motivation

Figure 1.1 shows the rapid development of new technologies and the related great demand for communication.

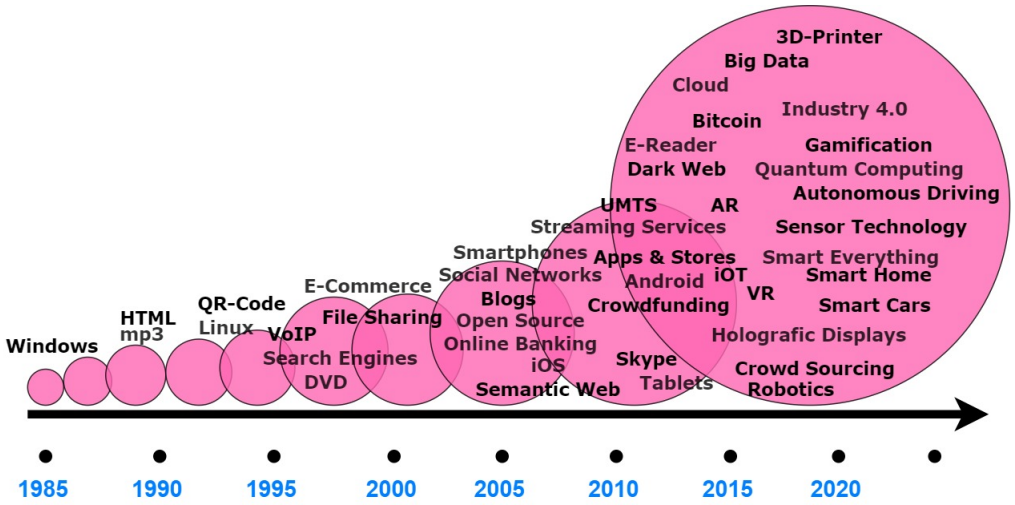


Figure 1.1: Technology Development Over the Last 35 Years [36, p.47]

The Internet as one possible supplier for this communication demand was one of the most disruptive technologies in history, and it could be able to help initiate further radical changes in our way of life, assuming the freedom to develop the necessary qualities to support future technologies exists. However, incremental changes and ad-hoc solutions instead of future-oriented, far reaching changes in the topology and mechanisms of the Internet led to an ossification [58] [4].
Network Virtualization (NV) [59] is a promising solution for this challenge by providing an abstraction of the network's underlying physical infrastructure [4, p.3]. With

efficient virtualization, physical infrastructures will be able to economically host many different Virtual Networks (VNs) which initiates healthy competition among network providers which in turn should reduce overall cost for realizing applications and services that leverage VN capabilities. It further enables experimentation with different topologies, routing and forwarding mechanisms personalised to their corresponding VNs [24, p.1]. VNs could rest upon multiple Quality of Service mechanisms which pave the way for newly emerging technologies like autonomous driving and Internet of Things related novelties. This thesis aims at investigating an evolutionary approach to support these network virtualization efforts. It consults genetics as a proven role model as a self-learning system and uses it for optimization - a major challenge in current network virtualization approaches [68, p.12]. This work joins first attempts to transfer mechanisms from the genetic domain into the domain of NV by providing a useful algorithm implementation and evaluating multiple noteworthy improvements.

1.2 Problem Description

NV describes the procedure of instantiating VNs on top of one or more physical networks, from here on referred to as Substrate Networks (SNs). Virtualization enables the coexistence of multiple VNs with widely varying characteristics. This new layout of network resources facilitates the creation of new network operator entities: Infrastructure Providers (InPs) which operate and manage the physical network and Service Providers (SPs) which translate applications and services into Virtual Network (VN)s and request the necessary resources from one or multiple InPs [23].
A qualitative implementation of this concept requires an effective utilization of network resources, more precisely an optimal allocation from the SN to multiple VNs. This allocation optimization problem is commonly known as the Virtual Network Embedding (VNE) problem [24]. With the dimensions of today's network infrastructures, the given amount of flowing data and the need to access operational performance information, we find ourselves in a highly complex and dynamic environment regarding the realization of this allocation. The complex computation of comprehensive solutions for the VNE problem challenges research and industry and slows down the transition into an Infrastructure as a Service (IaaS) model as the major instance for future network usage. Due to its NP-hard nature [3], sophisticated, often heuristic, optimization procedures must be used for the real-life application of NV to ensure profitability for InPs and thus enabling an IaaS model in the first place [13, p.2]. Widely varying network characteristics, requirements and constraints make the problem even more challenging. Due to the realization that NV can be used as a basis to investigate and realize new network architectures [14, p.1] and the possibility to overcome current shortcomings of the Internet, the effort is justified. Additionally, it guarantees innovation-friendly flexibility [52, p.4] and maximizes existing hardware utilization [24, p.1].

1.3 Goals of This Work

Genetic algorithms (GA) have been infrequently considered to tackle the VNE challenge, although they proved themselves very helpful in many kinds of complex global optimization problems [65, p.4]. This work aims at presenting GAs as suitable candidates for the VNE problem. For this purpose an overview over the VNE problem and the characteristics of its algorithms is presented and a comprehensive introduction into GAs and their usage in the VNE domain is given. With a basic implementation, this work further aims at providing a foundation for further investigations. It provides a common denominator for both domains, introducing multiple improvement strategies to give the reader a better understanding of the GA application in regards to complex problems.
The rest of this work is structured as follows: chapter 2 presents a formal description and background information about the VNE problem and the main characteristics of its algorithms. chapter 3 introduces GAs in detail with its main concepts and procedures. In chapter 4, related work of the application of GAs in the context of VNE is described and discussed. chapter 5 presents a basic VNE GA with specific details of its implementation. chapter 6 introduces advanced genetic operators which are evaluated in chapter 7. Finally, chapter 8 shortly summarizes this work and gives an outlook for future work directions.

2 VNE Basics

VNE describes the procedure of instantiating coexisting VNs on top of a shared physical infrastructure. The embedding follows the endeavour to partition a physical network, modelled as a graph, in the most optimal way by allocating the physical resources like CPU power and bandwidth capacities to the demanding VNs. [24]

2.1 VNE Example

The SNs as well as VNs are represented by a weighted (un-)directed graph consisting of nodes and edges as the depicted example in Figure 2.1. The substrate network

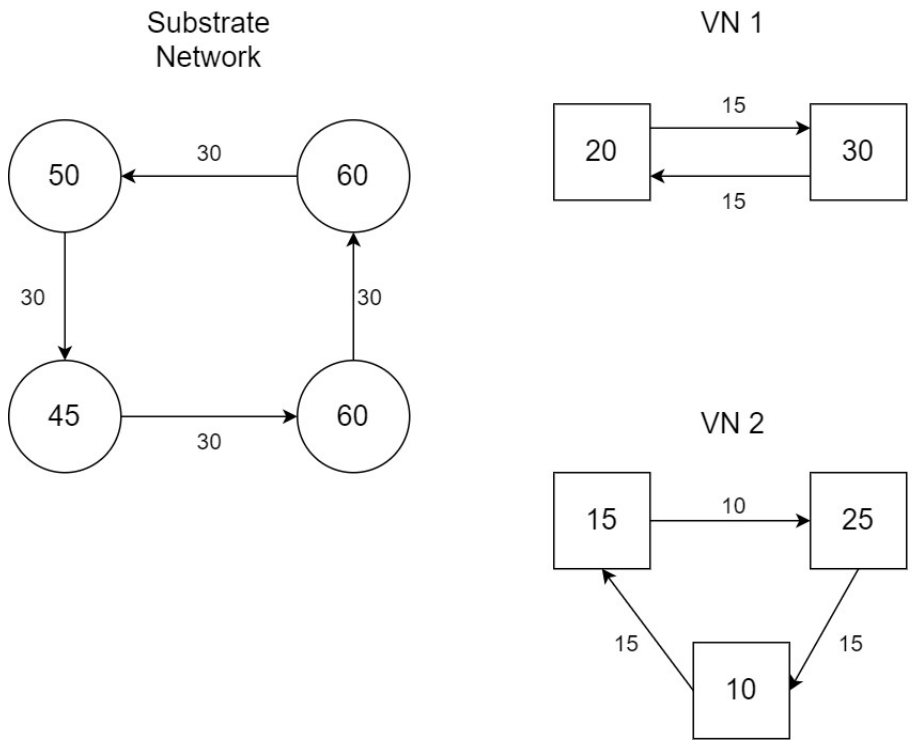


Figure 2.1: A Simple Substrate Network and Two Virtual Networks

consists of nodes with corresponding CPU capacities and links connecting the nodes which also have capacities in the form of bandwidth. The depicted virtual networks VN1 and VN2 have different topologies and different demands. VNE realizes a VN by allocating resources like CPU power and bandwidth to the demanding virtual nodes

and links while considering resource constraints. The implementing algorithm aims to make the best possible use of the substrate network, i.e. utilizing the resources in such a way that as many VNs as possible can be realised [68]. Figure 2.2 shows an example of the embedding of the previously depicted VN1. The virtual nodes with a demand of

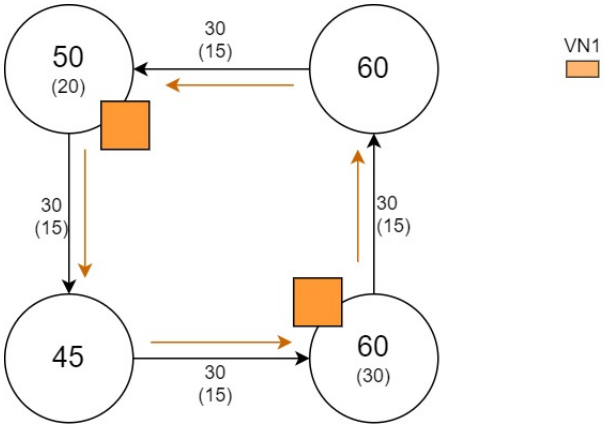


Figure 2.2: A SN with the Emedded VN1

20 and 30 are embedded into substrate nodes with available capacities and the virtual links are embedded into substrate paths. It is further assumed that routing does not consume any node capacities and that a virtual link can span multiple substrate links as depicted and described in [24]. The VN is synonymously referred to as a Virtual Network Request (VNR) in the subsequent chapters.

2.2 Formal Description

The SN as well as the VNR are modeled as a directed weighted graph. The following definitions are based on [19] and [43] and are adopted from [7]. The SN is denoted by $G^s := (N^s, L^s, C^s, B^s)$ and the VN is denoted by $G^v := (N^v, L^v, C^v, B^v)$. N^s defines a set of substrate nodes, where each physical node $n_i \in \mathrm{N}^s$ has an associated CPU capacity C_i^s. Also, N^v defines a set of virtual nodes with the associated CPU demand C_m^v of every virtual node, $n_m \in \mathrm{N}^v$. Furthermore, $L^s \subseteq \mathrm{N}^s \times \mathrm{N}^s$ is a set of substrate links, where each link $l_i \in \mathrm{L}^s$ and $l_{ij} = (n_i, n_j)$ has a corresponding link bandwidth capacity of B_{ij}^s. $L^v \subseteq \mathrm{N}^v \times \mathrm{N}^v$ is a set of virtual links $l_m \in \mathrm{L}^v$ and $l_{mn} = (n_m, n_n)$ with a corresponding link bandwidth demand of B_{mn}^v.
Nodes and links can have additional resources and demands like storage or latency, which are not considered in this thesis for simplicity but are typically taken into consideration for 'real world' implementations. The embedding of virtual networks maps the graph G^r onto the graph G^s and can be subdivided into two components: The virtual node mapping and the virtual link mapping [43, p.4]. The node mapping can be modeled as the node mapping function(2.1):

$$f_N : N^v \rightarrow N^s \tag{2.1}$$

which considers the following constraint:

$$\forall n_m^v \in \mathcal{N}^v : C_m^v \leq C^{s_N}_{f_N(N_m^v)}(t) \tag{2.2}$$

Equation 2.2 ensures, that the CPU demand of all virtual nodes is less or equal to the residual CPU capacity at time t of the corresponding physical node, considering all previous node mappings.
In addition, each virtual node of a VNR must be mapped to a different substrate node to ensure manageability and load balancing [13, p.5]. If every node could be mapped, the node mapping stage is rated successful.
The virtual link mapping can be modeled as the link mapping function(2.3):

$$f_L : L^v \rightarrow 2^{L^s} \backslash \emptyset \tag{2.3}$$

which considers the following constraint:

$$\forall l^v \in \mathcal{L}^v : \forall L_{ij}^s \in f_L(L_{mn}^v) : B_{mn}^v \leq B_{ij}^s(t) \tag{2.4}$$

Equation 2.4 ensures that the residual bandwidth capacity at a given time t of every embedding path in the substrate network supports the bandwidth demand of the corresponding virtual links. Note, that a virtual link is not limited to a single substrate link but rather can be mapped on a substrate path spanning multiple substrate links. If all paths fulfill equation 2.4, the link can be embedded and the link mapping stage is successful.
Furthermore, the revenue of a single VNR is defined as in [64, p.3]:

$$R(G^v) = \sum_{l^v \in L^V} B(l^v) + \sum_{n^v \in N^v} C(n^v) \tag{2.5}$$

Equation 2.5 represents the value for the embedding of the VNR from InP sight and summarizes all CPU and bandwidth demands of the respective VNR.
In order to represent the costs of a single VNR embedding, the following equation inspired by [44, p.2] is defined:

$$C(G^v) = \sum_{l^v \in L^v} \sum_{l^s \in L^s} f_{l^s}^{l^v} + \sum_{n^v \in N^v} C(n^v) \tag{2.6}$$

where $f_{l^s}^{l^v}$ defines the total amount of bandwidth assigned to the virtual link l^v from the substrate link l^s. In general, a VNE algorithm always aims to minimize the embedding cost defined in 2.6.
Two additional metrics are introduced which are commonly used to evaluate VNE algorithms. Because of varying VN sizes, it is appropriate to put the costs in relation to the generated revenue for each VNR in order to evaluate the efficient use of the substrate resources. Therefore the revenue to cost ratio is considered in this work

which is defined as:

$$RC(G^v) = \frac{R(G^v)}{C(G^v)} \tag{2.7}$$

Equation 2.7 represents the utilization of resources for a single VNR (G^v) and is commonly calculated across all accepted VNRs. It takes a value between 0 and 1, where 1 would represent an optimal embedding. Furthermore, the acceptance rate is introduced as a metric. This metric measures the percentage of VNRs that were accepted and as a consequence embedded by the algorithm. It is defined as:

$$AR(A_{VNE}) = \frac{\sum G_a^v}{\sum G_r^v} \tag{2.8}$$

Equation 2.8 represents the acceptance rate (AR) of a VNE algorithm (A_{VNE}), which equals the ratio of all accepted VNRs (G_a^v) to all requested VNRs (G_r^v).

2.3 Decomposition of the VNE Problem

As the authors of [19] state, most VNE algorithms divide the problem into two phases: the Virtual Node Mapping (VNoM) and the Virtual Link Mapping (VLiM). As such, the output of the first phase serves as input for the second phase. This division can result in poor performance regarding the acceptance rate or the R/C ratio, because each sub-problem is processed separately [24, p.6]. On the contrary, a coordination between the two phases is desirable but turns out to be more complex.
This fact results in fundamentally different algorithms which try to tackle the VNE problem: Some algorithms are explicitly uncoordinated and focus on VNoM and VLiM independently. They solve the VNE in two subsequent phases, like the algorithms presented in [64], [68] or [67]. E.g. the algorithm from the authors of [64] uses a greedy approach for its node mapping phase, in which virtual nodes with the highest demands are mapped to the substrate nodes with the highest capacities, followed by utilizing a shortest-path algorithm for the link mapping phase.
This node mapping approach based on a ranking can be extended by considering further aspects beyond merely the CPU capacity or demand of a node, e.g. outgoing edges as in [64] or even more sophisticated topology-based attributes as in [31]. This ranking is called Node Ranking (NR) and is a central mechanism that determines the order of the mapping of virtual nodes, as well as the allocation of virtual nodes to substrate nodes [44, p.1]. By considering not only CPU resources, an algorithm is implicitly coordinating both phases by attempting to optimize the subsequent link mapping phase within the node ranking scheme.
One of the most basic NR which goes beyond mere CPU capacity was introduced by the authors of [64], is denoted by CB and is defined as:

$$NR(u) = C(u) \times \sum_{l \in L(u)} B(l) \tag{2.9}$$

In this equation, the node ranking of node u equals the CPU capacity (or demand respectively) of this node, multiplied by the sum of all bandwidth capacities (or demands) of the links directly connected to that node. As an example, 2.3 shows two nodes with CPU capacities (or demands) and outgoing links with their corresponding bandwidth capacities.

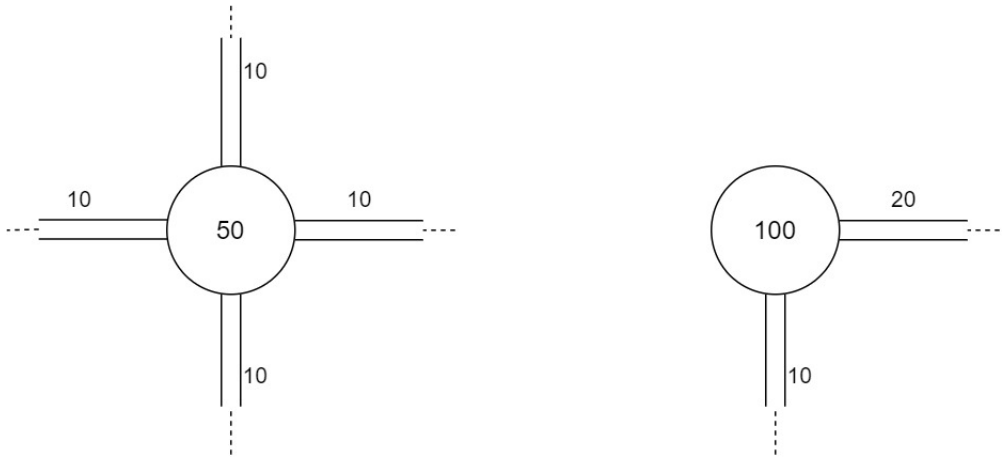


Figure 2.3: Two Nodes with Different Capacities and Outgoing Links

The value of the left node in Figure 2.3 is $50 \times (10 + 10 + 10 + 10)$ results in 2000, while the value of the right node in 2.3 is $100 \times (20 + 10)$ results in 3000. The right node is therefore ranked higher. This kind of ranking can be done for substrate nodes as well as virtual nodes.

Other algorithms try to utilize explicit coordination between VNoM and VLiM. This coordinated approach can be solved in one stage, where VNoM and VLiM are solved at the same time: After mapping the first two nodes, the link between all mapped nodes is inserted. Then, further nodes are embedded after which the corresponding links are inserted [17]. Alternatively, the embedding is solved in two stages as in [19], where the first stage results are used to optimize the results of the second stage [24, p.2]. The authors of [64] introduce a further concept called path splitting or multi-path. The authors that the limitation of a virtual link to a single substrate path restricts further mapping possibilities [64, p.4]. They propose an algorithm which considers splitting the virtual link demands into multiple flows, which softens the constraints on bandwidth. However, this thesis does not consider path splitting and utilizes single path mapping.

2.4 Classification of VNE algorithms

The VNE problem has been widely discussed in literature, and many algorithms have been proposed to address this issue [11] [12] [24]. This section will provide an overview of different classification schemes for VNE algorithms, as introduced by the authors of [24]. Fischer et al. characterize algorithms by their differing features, which will be briefly presented below.

Static or Dynamic
Static algorithms embed VNRs once and maintain the characteristics of the embedding, whereas dynamic algorithms can change the embedding afterwards. In reality, VNRs are dissolved after an indefinite period of time or change their characteristics during their lifetime, e.g. if an additional participant joins a network. The VNE problem is therefore realistically treated as an online problem where neither the VNRs nor their arrival time or their resolution time are known in advance. Static algorithms do not consider a re-embedding of already accepted VNRs and thus run the risk of fragmentation [24, p.3]. Dynamic algorithms can make subsequent adjustments to the VNs but consist of a considerably more complex implementation. [24, p.3f]

Centralized or Distributed
The authors, Fischer et al, further differentiate algorithms based upon the nature of the calculation instance implementation. Either the embedding is calculated centrally by a single instance or in a distributed manner across multiple instances. The advantages and disadvantages of both approaches are identical to other centralized or distributed applications: the central instance is constantly aware of the global status of the entire network. This information has a beneficial effect on the embeddings. However, it has a disadvantage due to a single point of failure, which would endanger the embedding of all subsequent VNRs in the event of a failure. If the embedding is calculated in a distributed manner, the embedding can be better scaled through better load balancing. However, this results in additional calculation effort, since information must be exchanged between the instances. [24, p.4]

Concise or Redundant
A further distinction concerns the amount of allocated resources, where concise and redundant embeddings have different implementation goals. Concise embeddings only take up as many resources as needed for an initial instantiation of the VN, whereas redundant embeddings allocate additional resources beyond the minimum. The advantage of redundant embeddings is that the VN can rely on a backup, and parts of the embedded network or the whole VN do not become inoperative in the event of a substrate failure. However, more resources are reserved in this scenario and can no longer be used for subsequent VN embeddings, with lowers the overall accepted VNs as a consequence. [24, p.4]

Exact or Heuristic

Finally, a distinction is made with regard to the solution strategy, which results in exact and heuristic algorithms. Exact algorithms provide optimal solutions for the VNE problem, but are impractical in larger problem instances due to their high computational effort, due to the NP-hard nature of the VNE problem [12, p.2]. Heuristic VNE algorithms on the other hand aim at a good balance between acceptance rate and runtime, which only leads to approximate solutions and possibly local instead of global optima [12, p.2]. While heuristic algorithms can get stuck in local optima, meta-heuristic approaches try to find a way to avoid this by better covering the search space for possible solutions and iteratively improving the found solutions [24, p.8f]. A strategy for such a meta-heuristic approach is introduced in the following chapter.

3 Genetic Algorithms

Genetic algorithms (GA) are general purpose metaheuristic search, optimization and machine learning algorithms. Based on a population approach, they are often used to minimize or maximize a particular function. They belong to the field of evolutionary algorithms which imitate and utilize methods and strategies inspired by nature. Other examples are Particle Swarm Optimization(PSO) and Ant Colony Optimization(ACO). [65]
This chapter serves as an overview of genetic algorithms and their basic operators. After a short introduction into the history of their development, the fundamental concepts are presented in section 3.1. Background information about the theory of GAs are presented in section 3.2 and the typical steps of GAs and a more detailed discussion of the basic operators are shown in section 3.3 and 3.4.

3.1 Key Concepts of GAs

GAs are adaptive systems and take over the biological processes of selection, recombination and reproduction, while randomly searching for the best solutions for their objective function. The main idea is that a population, which consists of many individuals, is able to evolve iteratively under a specific selection pressure towards the 'fittest' member [32, p.22]. The population makes use of a cooperative-competitive mechanism to search a large space from multiple points simultaneously [47, p.21].

A short outline of the development of GAs is now presented. A.S. Fraser, the author of [27], simulated reproduction, segregation and selection. H.J. Bremermann [9] and, later, J.H. Holland [33], developed the first adaptive systems in the context of automata, which were inspired by nature. Extensive contributions to the development and further understanding of GAs were made by the author of [33]. His theoretical investigations, focusing on crossover instead of mutation, identified crossover as a driving factor for GAs [38, p.1]. Finally, the author of [29] who was able to solve huge problems efficiently with GAs by using them for search and popularized the use of GAs for a broader audience [32, p.22f].
After a GA was applied to the network configuration problem by the authors of [20], for the selection of communication speeds of links, the result outperformed even designs created by experts and proved the applicability of GAs to this class of problems. Even if they are not the best choice for any kind of problem [32, p.23] they still are able to calculate acceptable to near optimal solutions for many real world problems, where other optimization methods fail [15, p.18]. As the authors of [38, p.1] state,

GAs are applicable to large scale combinatorial problems and are a good choice for solving maximization problems in an irregular or poorly understood environment. The following concepts are used by every GA:

The encoding mechanism is used to represent the problem which is to be resolved within the GA. An appropriate encoding of the optimization problems variables has to take place, while the exact encoding depends on the nature of the problem [55, p.3]. Typically, a binary or integer representation is used. As an example, the Traveling Salesperson Problem (TSP) would use cities as variables, which can be assigned ascending integer values, which in turn could be represented as binary strings, a typical representation used in GAs. Hence, GAs rather work with a representation of the problem's variables instead of using the variables themselves [22, p.22].

The chromosome represents the encoding of the problem under investigation and completely specifies a candidate solution. It is an array of values consisting of usually bits, but can also be strings or integers depending on the specific problem. If the problem has dimension N, then the chromosome also contains N variables in its array. We will adopt the definition of [32, p.30]:

$$chromosome(C_1) = [g_1, g_2, g_3, \ldots, g_N] \tag{3.1}$$

For example in the TSP, the individual variables $g_1, \ldots, g_N$, called genes, would represent cities. The sequence of cities displays the route which is taken by the salesperson. A single chromosome thus represents a possible solution to the problem, but does not necessarily have to be valid, i.e. respect constraints.

The population is a collection of individual chromosomes and represents the pool of possible coexisting solutions for the current generation. Therefore, it represents a parallel search with a large number of points in the search space. Assuming there are N genes in a chromosome and M chromosomes in a population, the latter can be described as a $M \times N$ matrix:

$$P = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_M \end{bmatrix} = \begin{bmatrix} g_1^1 & g_1^2 & \cdots & g_1^N \\ g_2^1 & g_2^2 & \cdots & g_2^N \\ \vdots & \vdots & \ddots & \vdots \\ g_M^1 & g_M^2 & \cdots & g_M^N \end{bmatrix} \tag{3.2}$$

The size of the population is usually user-specified and affects the scalability and performance of GAs significantly. As the authors of [53, p.2] state: *'[...] small population sizes might lead to premature convergence and yield substandard solutions. On*

the other hand, large population sizes lead to unnecessary expenditure of valuable computational time.' In GAs, a population is usually replaced by a subsequent generation in each iteration.

The Fitness Function, often referred to as cost function in optimization problems, mimics the selection pressure and can be an objective mathematical function or even a subjective function, carried out by human decision [53, p.1]. It is used to evaluate each chromosome according to its fitness, which is determined by how well the candidate solution optimizes the function. In the fitness function, it is decided which variables of the problem are the most important ones and how they are to be evaluated in relation to each other. Given the example of the TSP, the fitness function would calculate the travel expenses with the given sequence of cities.

3.2 Theoretical Background

The theory behind GAs is still under investigation and is not fully understood [15, p.29]. Holland [33] formulated his well-known '*Building Block Hypothesis*', which is the traditional approach for understanding the mechanics of GAs. As the author of [53, p.10] summarizes, on a very abstract level of understanding, GAs work by decomposition and reassembly of good Building Blocks (BB), i.e. good sub-assemblies of efficient solutions, to form high performance solutions in a parallel fashion. The explanation for this orientation towards global optima are so-called schemata, of which BBs are made of [38, p.35]. A schema can be interpreted as a template for the encoding, in the example below consisting of zeros, ones and *, which represents a wild card. E.g. the bit strings 0001 and 1101 both fit the schema **01. To illustrate this concept, a simple example in bit-strings is given, inspired by [38, p.36].

If given a set of arbitrary composed chromosomes, as depicted in the initial population below, the GA will implicitely identify good sub-assemblies (or BB) of those chromosomes.

Initial population	population $i+1$	population $i+2$	Final population
10101011	1010**1**011	10**1**01011	**1**0**1**01011
01001101	0100**1**101	01**1**01101	**1**1**1**01101
10100110	1010**1**110	10**1**01110	**1**0**1**01010
00110010	0011**1**010	00**1**11010	**1**0**1**01010
⋮	⋮	⋮	⋮
********	****1***	**1*1***	1*101***

While in the initial population no predominant schema is apparent, in the final solution a clear schema can be found: 1*101***. The schema can be seen as an orientation

for the search of the GA [38, p.35]. In the depicted example, the schema has four defined bits and therefore is of order four. The fitness of this schema is the average fitness of all strings that fit into the schema. The GA will evolve the schemata of the population hopefully towards well-performing higher order schemata, and combine multiple low-order schemata with each other through recombination, which will be further discussed in section 3.4.
The GA simultaneously evaluates all schemata present in the current generation. Here, each population of N individuals allows an estimate of the fitness of between 2^L and N^L schemata [38, p.36], called 'implicit parallelism' by Holland.
Holland explains how the schemata are passed on to the next generation in his mathematical formulation, called the '*Schema theorem*'. It is believed that short schemata with higher than average fitness occur exponentially more frequently in the subsequent generation than inferior schemata. However, this idea is not universally accepted, e.g. in [2], but still addresses selection and the negative aspects of BB disruption by crossover. The main point of criticism refers to the missing explanation of how crossover works to combine high performing individuals [46, p.94], which the authors of [46] try to explain with their royal road theorem.
A more intuitive idea, briefly discussed in [53, p.10], is that BBs '*exist in a kind of competitive market economy of ideas, and steps must be taken to ensure that the best ones grow [...] and the growth rate is neither too fast nor too slow.*' Those steps are implemented by appropriately setting the crossover probability and the selection pressure. Generally speaking, ' *[...] the identification and exchange of BBs is the critical path to innovative success.*' [53, p.12] regarding GAs.

3.3 Steps of GAs

In this section, the basic steps of a GA will be introduced with a short explanation of each. We will further investigate the specific operators of GAs in the next section. Figure 3.1 shows a flowchart of the typical GA application with the following steps.

After **Defining the Encoding**, as explained above, the **Fitness Function** has to be defined: As stated above, a fitness function needs to be introduced to the GA to guide its search. **Select GA Parameters:** The parameters of the GA need to be set. Usually this is done with constant variables and, at minimum, concerns the population size, the crossover or recombination rate and the mutation rate. The size of the population is typically set between 50 and 100, the crossover rate between 0.5 and 0.8, and the mutation rate typically 0.02 to 0.05 according to most publications. However, the authors of [38] suggest a mutation rate of 1/population-size and a crossover rate of 0.6 while referring to the work of [21].
Initialize Population: The first generation of the population is usually initialized randomly. Although there are known mechanisms to enhance the first generation [53, p.17], one of which will be revisited in chapter 5.
Evaluate Fitness of Each Individual: After the first generation is initialized, or

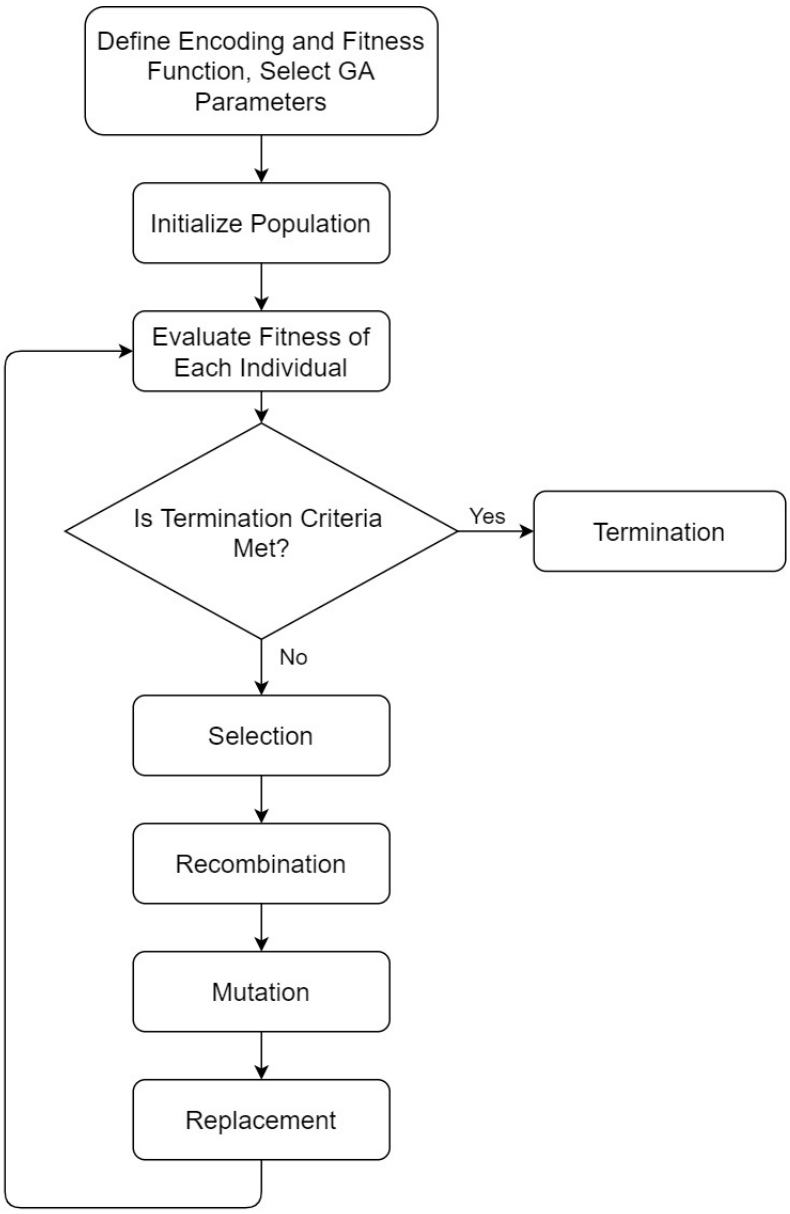


Figure 3.1: A Flowchart of the Basic Steps of a GA

after a subsequent generation is created, the fitness of every member of the population is evaluated by way of the fitness function. Based on the assigned value, i.e. the fitness, the next steps are determined for the individual. [53, p.2]

Termination Criteria: If this criteria is met, the GA terminates. This can be a preset number of iterations or a predefined fitness value that has to be reached or another parameter calculated during the search. Additional criteria for consideration can be found in [35].

Selection: This mechanism selects individuals from the population for further genetic processing, i.e. recombination and mutation. It ensures the survival of the fittest prin-

ciple by favouring the better performing chromosomes over weaker ones. This stage is crucial for setting the selection pressure under which the population evolves. [53, p.2ff]
Recombination: The subsequent generation is mainly generated by a recombination mechanism. There are multiple possible variations of recombination mechanisms, some of which will be discussed in the next section. Generally, this procedure exchanges multiple genes between two chromosomes, called parents, to form new individuals, called children. [53, p.2]
Mutation: The second important genetic mechanism alters individual genes randomly within a single chromosome. Both recombination and mutation are only applied with a certain probability. [53, p.2]
Replacement: This particular mechanism determines how and which individuals in a population are replaced by their successors. There are several procedures, the most common of which is generation-wide replacement, which replaces the entire generation in each iteration. Other possibilities contain a crowding strategy or steady-state replacement. In the crowding strategy an individual replaces the most similar parent or individual in the previous generation to preserve diversity [38, p.13]. The latter replaces a defined number of poorly performing individuals with newly-generated chromosomes in each iteration. [53, p.2]

3.4 Basic Operators of GAs

GAs mainly use three operators to evolve their population: selection, recombination and mutation. This section is dedicated to those basic operators.

Selection
The selection mechanism is summarized in two steps: (1) Identifying good solutions in a population and (2) assigning them a partner for the subsequent recombination. Depending on the selection mechanism, the individuals are taken directly from the current population or are chosen from a mating pool consisting of high performing individuals.
According to the authors of [29], selection mechanisms are either fitness proportionate or ordinal. The first describes randomly selecting either one or multiple candidates, which results in either roulette-wheel selection or stochastic universal selection (SUS) [5]. In roulette-wheel selection, the probability P of the ith chromosome of n candidates for being chosen as a parent can be described as:

$$P_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{3.3}$$

Here, the better performing individuals with their higher fitness values f_i have larger 'slot sizes' while the imaginary roulette wheel is spun. Instead of spinning the wheel multiple times with one pointer to get as many parents as necessary, it is possible to

spin it only once with an entire set of equally distributed pointers, thus selecting all parents in a single step, which results in SUS. The quantity of pointers reflects the number of parents wanted, and the number of pointers within a single slot reflects how many copies of that individual will be put into the mating pool.

Ordinal selection describes methods of differing approaches to selection. Elitist or truncation selection [48] is the approach in which only the highest-performing members of the population are chosen for subsequent genetic processing. In ranking selection [5], the entire population is sorted or ranked and an assignment function is used to assign fitness values to the individuals. Tournament selection [10] utilizes competition in its selection process, in which two or more individuals compete with each other and the winner is selected as parent for the subsequent genetic modifications. The number of individuals competing with each other, i.e. the size of the tournament, is proportional to the selection pressure. [30]

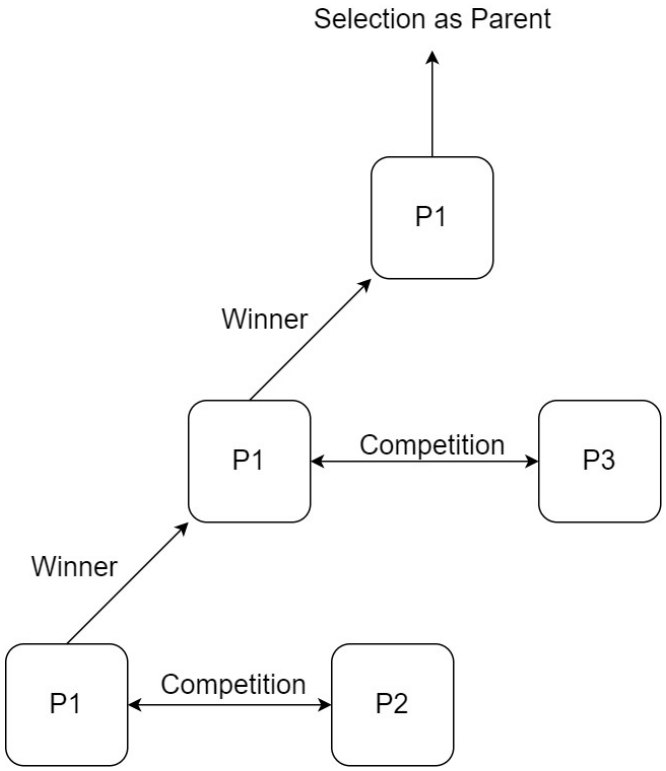


Figure 3.2: Tournament Selection with Three Participants P1-3

Other selection schemes, such as Boltzmann selection [28] or steady-state selection [56], are outside the scope of this thesis and will not be addressed here. In general, it can be said that the selection mechanism and, hence, the selection pressure is a fundamental component in GAs. If the selection pressure is too low, i.e. poorly performing individuals stay in the population for a long time, the convergence speed will be very slow. On the other hand, if the pressure is very high, then there is a higher chance of

premature convergence, i.e. the GA stops at a sub-optimal solution.
Several investigations into the impacts of the selection scheme have been conducted by various authors, e.g. [30]. As presented by the authors of [62] in their comparative study, tournament selection outperforms proportional selection concerning a steady convergence pressure, and the time complexity of repeated tournament selection outperforms linear ranking approaches due to the sorting component of the latter [30]. Deterministic schemes, like elitist selection, are most likely to result in achieving the highest fitness levels in the fewest number of iterations [42].

Recombination
After candidates are chosen as parents by the selection operator, the recombination (or crossover) takes place. Many recombination operators are described in literature, e.g. in [8], some of which will be introduced in this section. Determined to be one of the most important operators in GAs, they guide the search of the algorithm especially in the early stages by performing huge leaps in the search space towards good solutions [26, p.11]. However, this procedure is a balancing act between recombination opportunities and the disruption of potentially good solutions [60, p.8].
As described earlier, two parent chromosomes undergo recombination in the hope to create fitter children. Typically, this is done with only two parents, but multi-parent recombination [57] or gene pool wide recombination [49] can also be considered. This thesis will consider the two parent variant only.
Recombination between two individuals only takes place with a certain probability in GAs, called the crossover probability p_c. Failing this, the offsprings are identical to their parents, i.e. the parents are put directly into the next generation. A recent survey of recombination operators of GAs can be found in [60]. The authors state that the '*encoding type used in the GA is the major criteria for selecting the crossover*' [60, p.8] and come to the conclusion that new crossover mechanisms basically build upon old ones with additional alteration. For specific problems and their specific search spaces, a new or mixed crossover can be applied. In the following, a basic recombination mechanism is presented.

k-Point Crossover
Probably the easiest implementation of a recombination operator is the k-point, or more explicitly, the 1-Point Crossover (CO). In this operator, a random cut position within the string of the chromosome is calculated, and the genes of one side of the chromosomes are exchanged between the two parents, as depicted in the upper section of Figure 3.3. In 2-point crossover, as depicted in the lower section of Figure 3.3, two cut positions are randomly computed, and the entries between the positions are exchanged between the parents. This concept can be extended to an arbitrary amount of cut positions, where intersections are exchanged in the desired kind. [53, p.4f]

Mutation
Mutation describes the randomly and independently executed change of single genes within a chromosome with a very low probability, which is called the mutation prob-

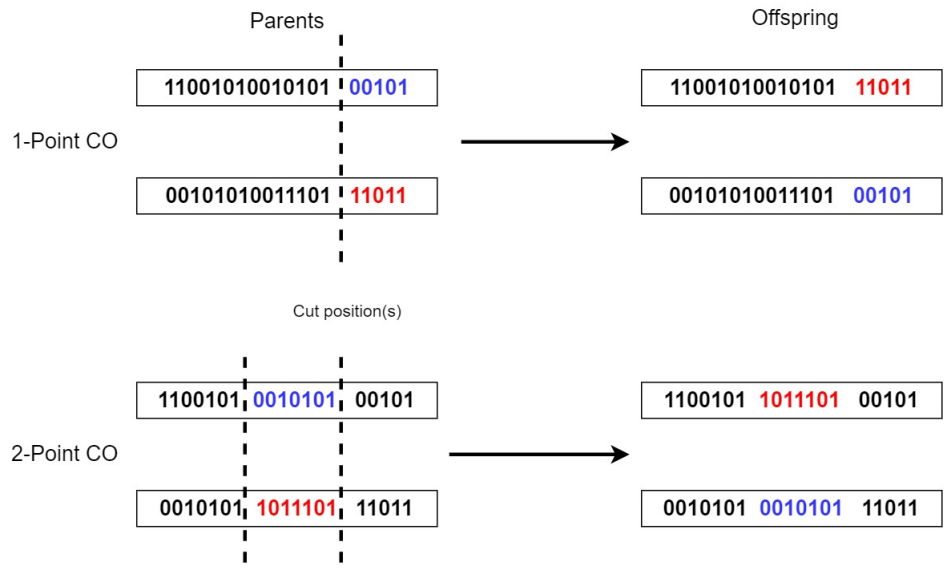


Figure 3.3: 1-Point and 2-Point Crossover Operators

ability p_m. In binary encoded chromosomes, this would be a simple bit-flip. In more problem specific variations, which have an encoding other than bit arrays, a mutation can be understood as a change of values, e.g. to an upper or lower bound. [53, p.7f] The purpose of mutation is maintaining or increasing diversity in a population [39, p.2]. This mechanism becomes more important in the latter stages of the GA, because the population will gradually lose its diversity. One could imagine an entire population made up of chromosomes in which each has only the value 1 at a certain position. This value, then, cannot be affected by recombination, but only by mutation [53, p.8]. By changing only small pieces of individuals, this mechanism explores the direct neighborhood of individuals in the search space. It also is responsible for the development of a chromosome from a very good solution to the optimum, and can be seen as a 'fine tuned' search mechanism, by which individuals can also escape local optima. [39, p.2] Basically, mutation can be classified in two categories: random and guided, where the latter is implemented deterministically using information from the population, in contrast to random selection, as implemented in the former. A comprehensive summary of different mutation approaches can be found in [39].
It is important to note that applying mutation to all chromosomes in the same manner without regard to their fitness is a major shortcoming. One would like to utilize a low probability factor to not disrupt good BBs, but if the individuals perform poorly, their mutation rate should be high to maintain exploration. [41]

The high impact of the latter two parameters on the performance of the GAs can be summarized as the following trade-offs: Raising the recombination (or crossover) probability also increases the reassembly of BBs, thus enhancing exploration, but also increases the probability for the disruption of high performance BBs, which in turn

deteriorates exploitation. Raising the mutation probability transforms the GA more into a random search walk but also opens the possibility to explore hidden neighboring peaks. [55, p.7]

3.5 Challenges and Advantages of GAs

To provide a comprehensive view on the subject, this section will describe the disadvantages and challenges of GAs. First of all, it must be mentioned that GAs are of a heuristic nature, and thus there is no guarantee that the algorithm finds an either optimal or acceptable solution. This often leads to a compromise between quality of solution and computational effort, which hence can be high. Another weak spot is the high number of control parameters. In principle, GAs are confronted with two optimization problems: (1) the problem the GA tries to solve, and (2) the choice of appropriate parameters as to population size, crossover and mutation rate. Further challenges include an effective problem representation, the formulation of an evaluation function, and the assurance that good solutions are not replaced by sub-optimal descendants or mutations prematurely. In general, GAs tend to pave the way to evolve into the neighborhood of the optimal solution, but they can have trouble finding the last few steps towards the global maximum. [6, p.9]
Despite the disadvantages just described, intensive research is still continuing in the field of GAs for several reasons. As Haupt et al. in [32, p.23] state, GAs do not require information about the environment and scour through the search space simultaneously from multiple points. This makes GAs parallel by nature, which makes them very well suited for parallel computers and approaches. Obviously, they provide multiple solutions instead of a single one. Moreover, the basic concept of GAs is easy to understand and to implement, making them accessible to a broad audience. This resulted in applications of GAs in many different domains like robotics, telecommunication, crypto analysis, biology and chemistry and the research of evolutionary optimization of NP-hard problems is still an ongoing quest with promising results, [37, p.2] and making GAs a good candidate for the VNE problem.

4 Related Work

The application of GAs in the context of VNE is quite new, with the first application in 2012 by Mi et al. [44], providing the research community with a basic GA for the VNE problem. The authors use two different versions of node rankings and utilize a separate algorithm for the initialization of their population. In general, the authors seem to put exploration before exploitation by using genetic operators and mechanisms, which do not put a high selection pressure onto their candidate solutions, but are re-initializing new points in the search space continuously.
In terms of the node ranking scheme, the authors support the version utilizing the CB ranking method to be more effective than the investigated Random Walk ranking method [17]. In their evaluation, the authors compare different evolutionary-based algorithms, and come to the conclusion that GA based algorithms outperform PSO-based algorithms regarding acceptance ratio, average revenue and R/C ratio. However, the authors neglected to investigate additional important metrics, like running time, and did not investigate other genetic operator approaches.
A comprehensive performance evaluation by Chang et al. [16] compared the classical versions of multiple EA based algorithms, namely Particle Swarm Optimization(PSO), Ant Colony Optimization(ACO), and GAs, and further examined the effect of the node ranking scheme on the algorithm performance. Their results indicate that the features of an AI technique have a greater impact on the examined objective, namely average revenue, than the node ranking scheme. They further conclude that GA-based algorithms perform better than PSO based algorithms, in terms of R/C ratio, acceptance ratio, and average revenue. While ACO based algorithms outperform GA-based algorithms in small and medium sized scenarios in terms of average revenue and acceptance ratio, GA-based algorithms are able to keep up in large scenarios in terms of R/C ratio. It should be noted that the GA used in the evaluation is identical to the GA used by Mi et al. and is, therefore, a very simple variant of all GAs. A comparison of more sophisticated versions of the mentioned algorithms is still pending.
A memetic algorithm, a combination of a population-based approach in combination with local improvement techniques, is introduced by Inführ et al. in 2014 [34], which examines the implementation of different versions of uniform crossover and different hybridization techniques, as well as different representation schemes. The authors aimed to investigate, among other things, whether the time spent for local improvement techniques is well spent. Their results are as follows: Local improvement techniques are beneficial in small instances of virtual embeddings, but further GA iterations are better suited to find acceptable solutions for large problem instances.They also consider local improvement essential for high loads of VNRs.
Liu et al. [40] introduced a mixed GA in 2016, which utilizes the simplex method

to escape from local optima. Based on the implementation of [44], a modified GA approach has been undertaken in 2017 by Zhang et al. [66] , which mainly focuses to improve two aspects: (1) the mutation operation, where the authors mutated nodes towards nodes that are located closer to the already mapped nodes, and (2) a further improvement method, which remaps the least desired single node mapping of a given mapping solution. Their simulation results show an increased long term average revenue and acceptance ratio of VNRs using their GA-based algorithm, in comparison to two D-VINE algorithms [18], current state of the art. However, only small improvements, mainly regarding the genetic operators, are made by the authors of the mentioned work. Known enhancement techniques for GAs, like additional parallelization and hybridization, are only superficially taken into account.
Nguyen et al. [51] recently proposed their intelligent parallel algorithm, which is a GA-based link embedding algorithm for an online environment. By finding multiple possible solutions on distributed machines, they demonstrated results in terms of acceptance ratio, and cutting down the average cost of accepting VNRs. Their valuable contribution of their distributed approach and master-slave implementation makes effective use of lowered execution time. With one exception, the authors restrict their evaluations to only the basic implementations of genetic operators, thereby failing to investigate newly emerging aspects of GAs or taking into consideration different genetic operators.

To overcome some of the shortcomings in current literature, this thesis presents a GA which combines aspects of the previously discussed work and investigates further improvements in terms of acceptance rate, R/C ratio and running time. Differently composed GAs in the context of the VNE problem and further improvement approaches are investigated on the basis of the algorithm introduced in the following chapter.

5 The Basic VNE Genetic Algorithm

Successful applications of GAs to multiple NP-hard combinatorial optimization problems make the application of GAs to the VNE problem a promising approach. This chapter introduces the Basic VNE GA, which is able to solve instances of the VNE problem and is inspired by [44].

5.1 Algorithm Overview

To understand the mechanism of the following algorithm, an overview is presented in Figure 5.1. The Basic VNE GA takes a network stack consisting of the substrate network and an undetermined number of VNRs as input. The algorithm terminates if all VNRs were processed. For every VNR, the GA with predefined parameters is executed, which then returns a candidate solution in a form of a node mapping solution for the VNR. This solution is then embedded while using a k-shortest path algorithm for the link mapping stage. After all VNRs have been processed, the network stack containing the substrate network with all assigned resources, is given back as output.

The Basic VNE GA is an uncoordinated two-stage algorithm. It uses a probabilistic greedy approach to initialize a population of node mapping solutions. As mentioned in chapter 3, the Basic VNE GA can be classified as meta-heuristic. It is evaluated in a static environment and is computed, using a single, centralized instance. Furthermore, it assigns resources via a concise approach. The used k-shortest path algorithm is the version introduced by Yen [63]. Based on Dijkstra's algorithm, it aims to find the shortest path between two nodes in a graph topology and does not support loops. The difference to Dijkstra stems from the fact that the k-shortest path algorithm is allowed to fail due to constraints that are not met. If so, the algorithm proceeds to find the second-shortest path, the third-shortest path and so on. It is the norm to limit the algorithm with an upper bound to give the algorithm an appropriate termination criteria. This termination bound is set to a value of four in this work.

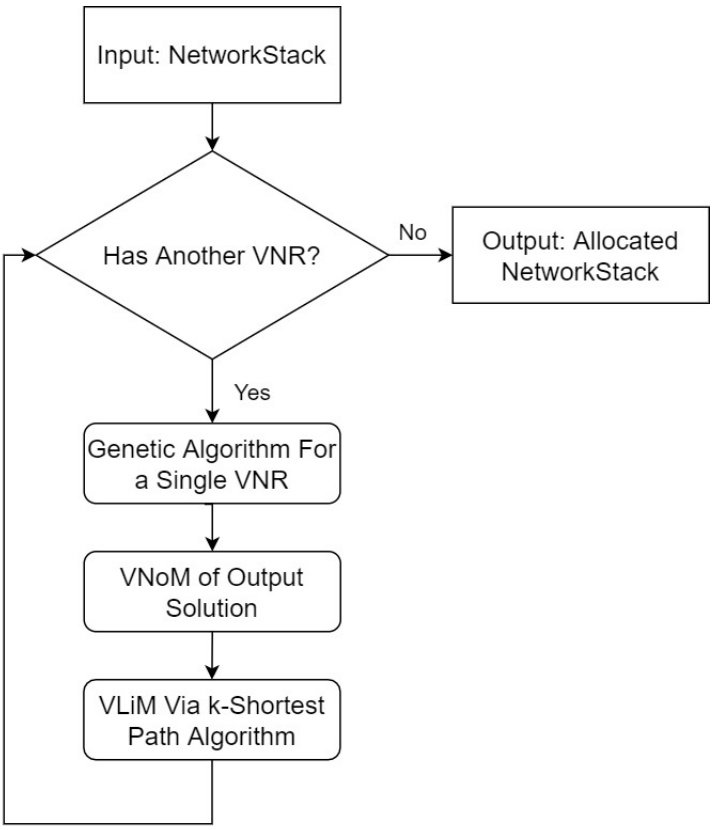


Figure 5.1: A Flowchart of the Basic VNE GA

5.2 Algorithm Functionality

The following description focuses on the implementation of the Basic VNE GA and follows the sequence depicted in Figure 3.1.

The Encoding in this algorithm describes a candidate solution for the node mapping stage of a single VNR. The number of genes in a chromosome is determined by the size of the VNR, more precisely the number of virtual nodes of the VNR. The genes are encoded as integer values, representing the IDs of substrate nodes, to which the virtual nodes should be mapped. This is called the host vector and is defined as:

$$H_i = (h_i^1, h_i^2, \ldots, h_i^{|N^v|}) \tag{5.1}$$

Here, h_i^k represents the physical node which hosts the k^{th} virtual node in the i^{th} chromosome. An example of a chromosome representing a VNR with eight virtual nodes is given in Table 5.1. In this example the first virtual node of the VNR is hosted by the physical node with ID 1. The second virtual node is hosted by the physical node with ID 4 and so on.

1	4	7	2	9	11	3	8

Table 5.1: An Example Chromosome for a VNR with Eight Virtual Nodes

Duplicates in this host vector are prohibited, otherwise multiple virtual nodes of the same VNR would get mapped onto the same substrate node which would violate the constraint mentioned in section 2. *The Fitness function* aims to minimize the embedding cost, which was already introduced in chapter 2 and is defined in equation 2.6. *The GA parameters*, more precisely the crossover probability and the mutation probability used in this variant, are inspired by [44] and based on [20]. The parameters can be taken from the following table.

Generations:	10
Population Size:	10
Crossover Probability:	0.8
Mutation Probability:	0.05

Table 5.2: Basic VNE GA Parameters and Their Corresponding Settings

The generation size of ten was chosen after first evaluations of the algorithm showed that more improvements regarding the embedding cost have been achieved in the early generations. This can be seen in Table 5.2, which shows three distinct VNs and the development of the respective minimal embedding cost over the generations. Regarding the first and the third VN, most improvements occurred before the tenth generation. In general, this work attempts to improve the algorithm in the early generations, for reasons of algorithm efficiency. However, the depicted charts also show that further improvements can be made after the tenth generation, and a complete lack of focus on those late generations can lead to performance losses. Nevertheless, the author of this thesis decided to focus on early generations, also to keep evaluation and test runs short in this early development stage.

The following algorithm is used to process a single VNR and takes the VNR and the SN as input. It results in either a valid node mapping solution or returns a notification of an unsuccessful try. The mechanism is described in the following (Algorithm 1) and an explanation will be given afterwards.

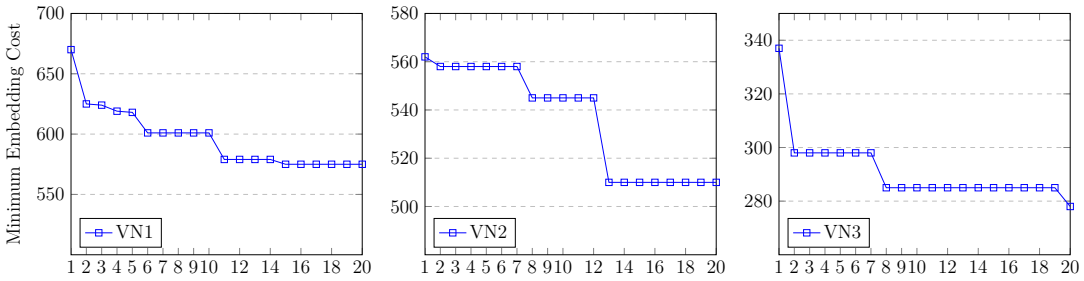


Table 5.3: Three Different VNs and their Minimal Embedding Cost Per Generation

Algorithm 1: Node Mapping Algorithm

Result: Valid embedding or error notification **Input:** G_v, G_s

```
1  set gBest = ∞
2  while popSize < popSize_set do
3  |   create new chromosome C_i and add it to pop_1;
4  |   C_i = Initialization(C_i, G_v, G_s);
5  |   feasibilityCheck(C_i);
6  |   computeFitness(C_i);
7  end
8  while iterations < iterations_set do
9  |   generate pop_{i+1} through Selection, Reinitialization, Crossover and
   |    Mutation ;
10 |   for Each chromosome C_{i+1} in pop_{i+1} do
11 |   |   feasibilityCheck(C_{i+1});
12 |   |   computeFitness(C_{i+1});
13 |   |   if gBest < fitness(C_{i+1}) then
14 |   |   |   gBest = C_{i+1};
15 |   |   else
16 |   |   |   continue;
17 |   |   end
18 |   end
19 end
20 if gBest ≠ ∞ then
21 |   return gBest;
22 else
23 |   return error notification;
24 end
25
```

The algorithm first initializes a global best embedding, denoted as $gBest$, with infinity(line 1). It then initializes the first population pop_1 with chromosomes until the

predefined population size is reached(lines 2-7) by creating chromosome objects(line 3) and applying the Initialization Algorithm(line 4) to them. The Initialization Algorithm (Algorithm 2) takes a single Chromosome C_i with the corresponding host vector H_i (which is empty at this time), the VNR G_v and the SN G_s as input. This mechanism is inspired by the implementation by [44] and is described below.

Algorithm 2: Initialization Algorithm

Result: Filled C_i **Input:** C_i, G_v, G_s

```
1  for Each physical node v in G_s do
2  |  Set v = untouched;
3  end
4  Calculate node rank(NR) for all virtual nodes;
5  Calculate node rank for all physical nodes;
6  Sort virtual nodes in decreasing order;
7  while !(all virtual nodes mapped) do
8  |  Calculate candidate list CL(u) for highest ranking virtual node;
9  |  Select one physical node v from CL as in equation 5.2 as a host for u;
10 |  Set v = touched;
11 end
12 Return Filled C_i;
```

The Initialization Algorithm first assigns all physical nodes the attribute untouched (lines 1-3), which flags the physical node available as a host candidate. After that, the node ranks of all virtual nodes are calculated using the node ranking defined in equation 2.7. The same is done with all physical nodes (lines 4-5). After sorting the virtual nodes according to their node rank in decreasing order (line 6), a candidate list (CL) consisting of potential host nodes is created for every virtual node, starting at the highest ranking virtual node. The candidate host nodes must fulfill the CPU constraint and have to be flagged untouched. The probability for selecting a node v from the CL is defined as:

$$P_v = \frac{NR(v)}{\sum_{v \in CL(u)} NR(v)} \tag{5.2}$$

This selection resembles the roulette wheel selection in section 3. The selected node is then set to touched(line 10) and can therefore no longer serve as a potential host node for the chromosome to be initialised. This procedure is repeated until the host vector is filled completely (lines 7-11). Finally, the chromosome with it's filled host vector is returned.
After creating the chromosomes, the Node Mapping Algorithm (Algorithm 1) checks every chromosome upon it's feasibility which is done via the k-shortest path algorithm. This path algorithm checks whether all virtual links of the VNR find their hosted physical paths using the node mapping encoded in the corresponding chromosome and if successful, the chromosome is flagged as feasible. While doing the feasibility check, the

fitness of the chromosome, i.e. the cost of the embedding, is calculated simultaneously as defined in equation 2.6 and is then assigned to the chromosome(lines 5-6).
At line 8, the Node Mapping Algorithm starts to iterate through a predefined number of generations. Line 9 will now be discussed in more detail. A schematic representation is given in Figure 5.2. Figure 5.2 shows the creation of a new generation in

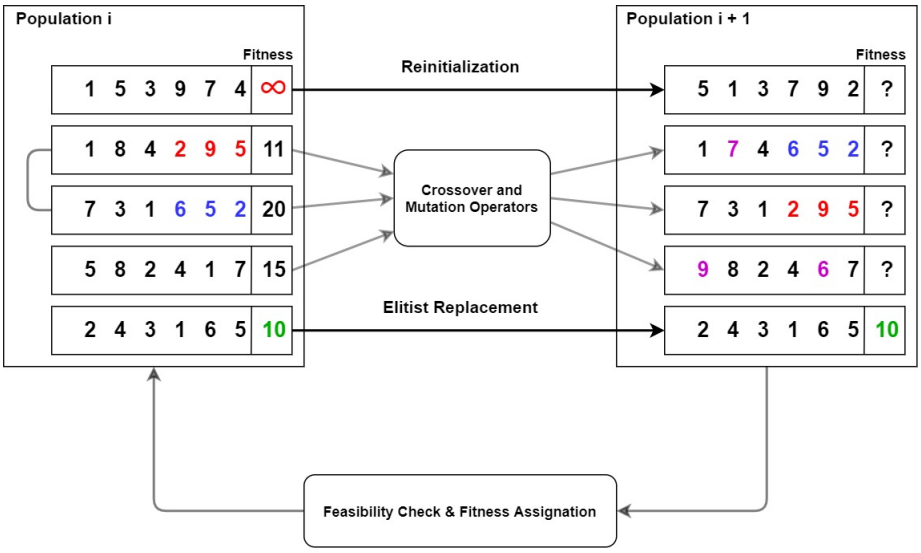


Figure 5.2: Schematic Creation of a new Generation in the Basic VNE GA

the Basic VNE GA. On the left an initial *Population i* is depicted consisting of five unique chromosomes with different fitness values. Note that one chromosome has a fitness value of infinity assigned, which results from the feasibility check in the previous iteration or the initialization. The check revealed that the encoded node mapping does not support a successful link mapping and the chromosome is therefore flagged as infeasible. All infeasible chromosomes become newly initialized once again by the Initialization Algorithm and are then put into the subsequent generation. Furthermore an elitist mechanism is performed, which chooses the chromosome with the lowest cost function (i.e. the highest fitness), which is 10 in this example, and puts it directly into the next generation without any alteration by genetic operators. The other feasible chromosomes are paired in a random fashion for a potential crossover. In this example the second and third chromosome are chosen and a simple 1-Point Crossover as described in chapter 3 is applied with the probability p_m. To ensure that every virtual node is hosted by an individual substrate node, the entries in the host vector have to be unique. Applying the crossover operator can violate this constraint, therefore a repair mechanism is needed to ensure the unique occurrence of host node IDs. This repair mechanism is presented in Figure 5.3.

The repair mechanism for the crossover operator consists of the following series of steps: It first detects conflicting genes after the crossover operator has been applied

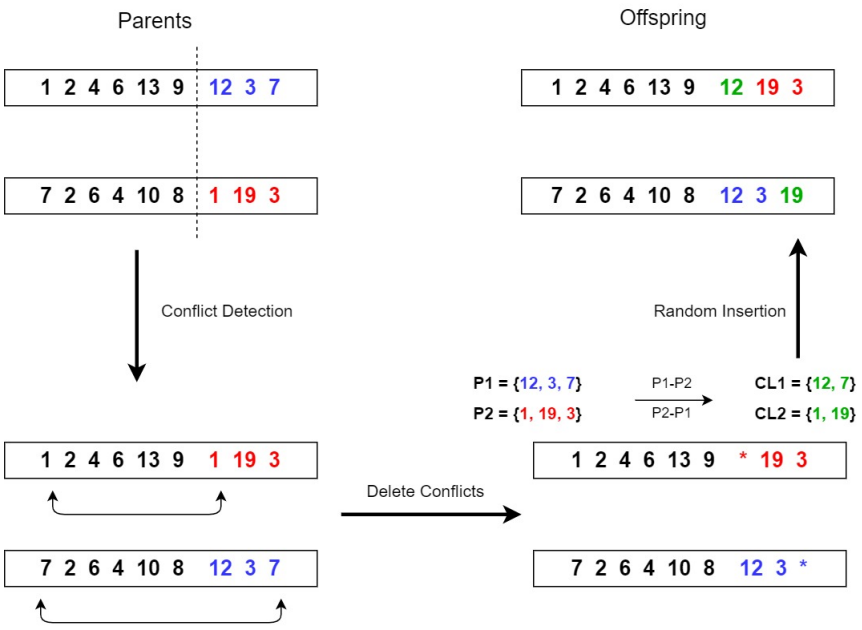


Figure 5.3: Offspring Creation by 1-Point Crossover with a Repair Mechanism

and deletes them. It then creates candidate lists with possible gap filler nodes for both child chromosomes, denoted as CL1 for the first and CL2 for the second child chromosome. To generate these lists, the already crossed-over genes are first put into separate lists P1 and P2. P1 represents the list consisting of genes obtained from the first parent and P2 represents the list consisting of genes obtained from the other parent. CL1 is now generated by *'subtracting'* the lists P1 and P2. CL1 now only consists of nodes which are guaranteed not to be part of the first offspring already. The same procedure is done for CL2, after which one gene is chosen from the candidate list in a random fashion to fill the gap in the respective chromosome.
After this recombination step, the **mutation** operator is applied with the probability p_m on every gene in every chromosome, regardless of whether the crossover operator is applied. The basic mutation operator is schematically illustrated in Figure 5.4.

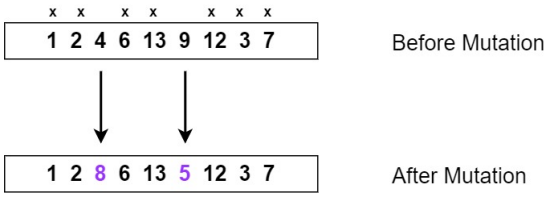


Figure 5.4: The Application of Simple Mutation

The mechanism in Figure 5.4 first checks for every gene in a chromosome to determine if the mutation operator should be applied. If this is the case, it then creates a candi-

date list of substrate nodes which satisfy the CPU constraint and are not yet contained in the host vector. Then, a host node is chosen in a random manner to replace the gen in the chromosome and the ID of the new host node is then written into the gene. In the depicted example, the third and the sixth entry were selected for mutation, and their entries got replaced with new unique IDs. The second mutation in gene six could have been mutated to ID four as well, assuming it fulfills the CPU constraint. If no other suitable candidate node could be found, the mutation is discarded.
After creating a new generation and replacing the old one, the feasibility and the fitness of each member of the population are checked and assigned, as shown in Figure 5.2, which represents the lines 11 and 12 of the Node Mapping Algorithm. If any newly calculated fitness is better than the prior initialized global best *gBest*, the corresponding chromosome is set as a new global best (lines 21-22). Finally the Node Mapping Algorithm returns the global best chromosome, if there is any. Otherwise a failure notification will be returned.
In summary the Basic VNE Genetic Algorithm uses a 1-Point Crossover mechanism, a simple mutation operator, an elitist replacement strategy and random selection. It further utilizes a dedicated initialization method and serves as a standard reference approach in the evaluation.

6 Enhancement Investigations

There are many sources discussing potential approaches for improving GAs [53] [32] [50]. With many different genetic operators and dedicated approaches and even more possible combinations, an investigation of all of them is beyond the scope of this thesis. In chapter 4 some of the shortcomings in using GAs in the context of VNE were discussed, one of which is the missing evaluation of multiple different genetic operators and their impact on different metrics. This chapter will introduce the operators and mechanisms which will be evaluated in chapter 7.

6.1 Recombination

Besides the 1-Point Crossover, introduced in section 3.4, the Uniform Crossover procedure is now introduced. With this operator, a descendant takes over the respective parent gene at a certain probability. This probability is set to 50% by default. However, different variations can be implemented by adjusting the probability of taking one gene or the other. Spears et al. [54, p.3] state for example, that an adjustment to 0.1 is *'less disruptive (overall) than 2-point crossover and has no defining length bias'*, thus has no fixed limits but the possibility to take over whole sections from a parent. The procedure with a 50% probability is shown schematically in Figure 6.1.
As can be seen in Figure 6.1, each gene within the offspring chromosome is inherited by 50% from the respective parent chromosome. An identical copy of one of the parents is therefore theoretically possible, however, this probability decreases dramatically with increasing length of the chromosome. While 1-Point Crossover has a very low probability of breaking well-performing BBs due to their one-time disruption, Uniform Crossover has a very high probability for the disruption of BBs. However, with Uniform Crossover, it is very easy to produce significantly more diverse offspring than with 1-Point Crossover and, as such, results in greater emphasis on exploration. It is important to note that a simple implementation of this operator is not sufficient in the case of the VNE problem, because it can lead to a violation of the property of unique genes - just like 1-Point Crossover. Therefore, a repair mechanism must also be applied to prevent invalid offspring. This is shown in the lower part of Figure 6.1, where the last gene can be adopted by neither of the parents because both entries are already contained in the descendant. To repair this gap, the first entry of the first parent which is not yet present in the offspring is used, which is eight in this example. Furthermore, in this variant, only a single offspring chromosome is formed, whereas

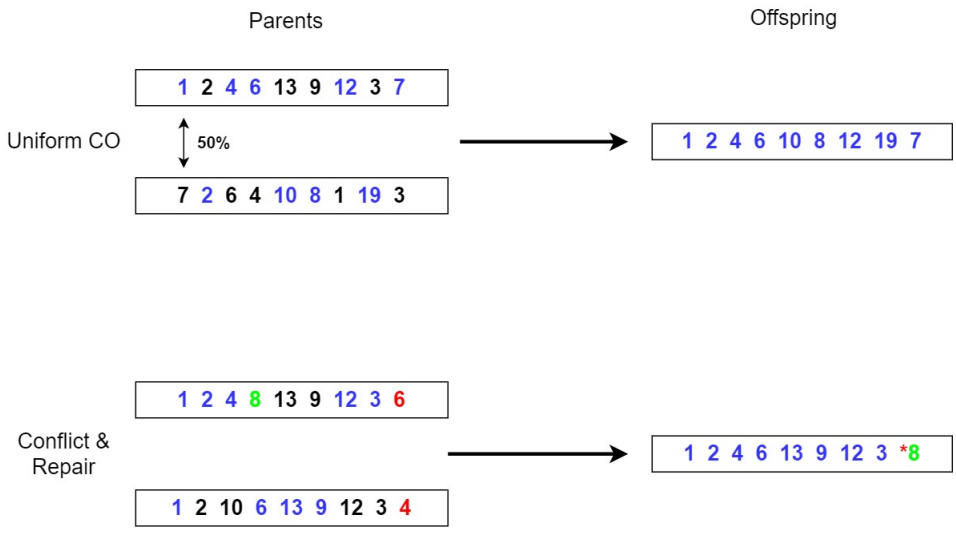


Figure 6.1: Uniform Crossover Operator and it's Conflict Repair Mechanism

in literature often a second offspring is formed simultaneously as complementary to the first. Here, for the formation of a second offspring the parents are swapped and the procedure is performed again. In chapter 7, this mechanism is once applied with a probability of 50% to inherit a gene from the first parent, and once with a probability of 10%, denoted as Uniform0.5 and Uniform0.1.

6.2 Mutation

As alternatives for the Simple Mutation introduced in Section 5.2, two further variants are presented now. The first variant is called Inversion Mutation [25], it's procedure can be seen in Figure 6.2.

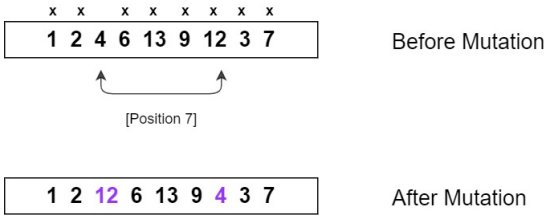


Figure 6.2: Inversion Mutation Operator

This mutation operator, applied with the probability p_m as all other mutations, swaps two genes within a single chromosome. In contrary to the Simple Mutation mechanism, two genes are modified in this variant. But instead of adding an additional node

into the mapping, this operator applies a remapping within a single node mapping solution. If a node is chosen to mutate, a second position unequal to the initial node position within the chromosome is randomly calculated as a swap partner. In the depicted example, the mutation is applied to the third gene and position seven was selected as a swap partner. This partner has to have enough CPU capacity to host the initial node instead, and the initial node has to have enough CPU capacity to host the swap partner. If the capacities are not sufficient, a new swap partner is calculated. This is done for a maximum of five attempts.
Another investigated variant is the so-called Neighbour Mutation. Its unique feature is, that it only targets nodes that are in the direct neighborhood of the remaining nodes of a node mapping solution and also fulfill the CPU requirement. The neighbourhood of a node is defined as a set of nodes which are directly connected to this node via a link. This principle is illustrated in Figure 6.3. It shows a SN with a

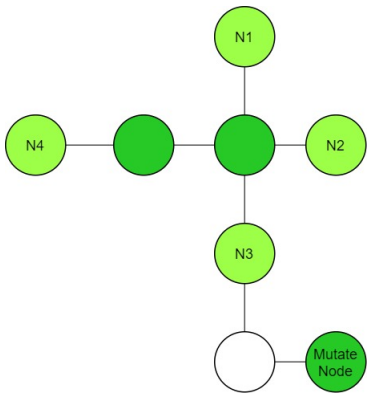


Figure 6.3: Potential Nodes for the Neighbour Mutation

potential node mapping solution of a VNR consisting of dark green nodes. Applying Neighbour Mutation to the node at the bottom would result in a remapping of this node to one of the neighbours of the remaining nodes, denoted as N1-4. The neighbour nodes considered must fulfill the CPU constraint.

6.3 Replacement

The Basic VNE GA replaces the entire previous generation of chromosomes with the newly computed ones, except for the best performing chromosomes from the previous generation, which is directly copied by applying the Elitist Replacement Strategy. Alternatively, the Truncation Replacement mechanism, inspired by the Truncation Selection [48], can be used instead. In Truncation Replacement, after a new generation has been formed by the genetic operators, a percentage of the best performing individuals from both generations is transferred to the final population. The mechanism,

therefore, uses an intermediate step before each further iteration. This thesis uses a percentage of 33% and fills the remaining two thirds with randomly chosen chromosomes from the new generation created by crossover and mutation. Contrary to the Elitist Replacement in the Basic VNE GA, the best chromosome of a population is now subject to recombination and mutation, as well. The procedure is schematically depicted in Figure 6.4, where the three best performing chromosomes from both populations are transferred to a final population. This is done in every iteration of the algorithm. Note that the already chosen chromosome depicted in the middle of Figure 6.4 can also be chosen for filling up the final population.

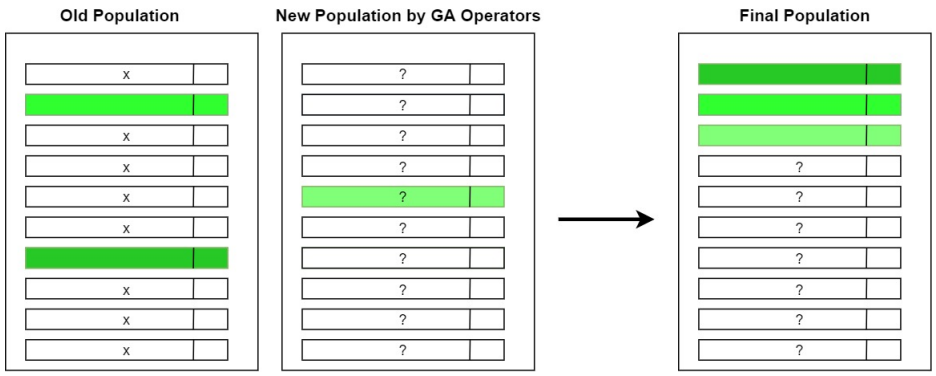


Figure 6.4: Truncation Replacement of Three Individuals in One Iteration

6.4 Selection

The Basic VNE GA does not pursue any kind of advanced selection mechanism. It pairs parent chromosomes in a random fashion without any consideration of their performance. This does not put any selection pressure on the population and therefore does not support faster convergence. As a possible improvement to random selection, Tournament Selection [10] will be applied. This mechanism was introduced in section 3.4 and is implemented with two participants, which is called a binary tournament. Both contestants are randomly chosen from the population, after which their respective fitness value is compared. The individual with the better fitness is then used for the subsequent genetic operators.

6.5 Termination

The termination criteria for the Basic VNE GA solely consists of the predefined number of generations, regardless of the performance of the algorithm or the quality of

the solution reached after each iteration. Therefore, an additional termination criteria is introduced, which prevents further unnecessary iterations with a simple check after each iteration, to determine weather the algorithm is able to embed the found mapping solution with 110% of the VN's minimal embedding cost, or less. In other words, it is determined that an embedding with additional 10% cost to the minimal cost is sufficient and the algorithm terminates in that case. The minimal embedding cost represents an optimal embedding with no utilization of additional resources than absolutely necessary, and equals the revenue of the respective VNR. Using the definitions in chapter 2 the termination criteria is defined as:

$$C_i(G^v) \leq 1.1 * R(G^v) \tag{6.1}$$

Equation 6.1 defines a termination of the algorithm, if the overall cost of a found solution i is less or equal than the revenue of the respective VNR multiplied with the value 1.1.

7 Evaluation

The implementation of the Basic VNE GA as well as the implementations of all extensions introduced in chapter 6 were done in ALEVIN [1], an open source java project that provides a complete documentation and a helpful framework for implementing and testing VNE algorithms. It supports the implementation presented here through provided network classes and methods as well as generators for various scenarios.

7.1 Test Environment

The algorithm described in section 5 and all further variants are tested in 20 different scenarios or network stacks respectively. These scenarios are mainly divided into two environments: One environment consists of a substrate network with 30 nodes and ten (pseudo-)randomly generated VNs with 2-10 nodes, and is referred to as the small testing environment. The second environment consists of a substrate network with 80 nodes and also ten (pseudo-)randomly generated VNs with 6-13 nodes. It is referred to as the big testing environment.
All networks are generated with a Waxman topology generator [61] which initializes a preset number of nodes and connects each node with a certain probability to each other and thereby creates the links. This probability is set to 0.5 for the SNs and to 0.7 for the VNs. Every node in the SN has a CPU capacity of 50-100 and every link in the SN has a bandwidth capacity of 50-100, while the nodes of the VN have a CPU demand of 20-50, and the respective links have a bandwidth demand of 20-50. The exact numbers for capacities and demands were generated in an equally distributed manner within the bounds described above.

The evaluation mainly focuses on metrics introduced in Section 2.1, namely R/C ratio and acceptance rate. It further investigates the average solving time of the algorithm with the respective features and counts early terminations as defined in equation 6.1.

7.2 Evaluation Results

Recombination Operators
In the following charts, the impact of the different recombination operators presented in chapter 6 on the performance of the algorithm introduced in chapter 4 is evaluated.

As shown in Table 7.1, none of the recombination operators performed significantly

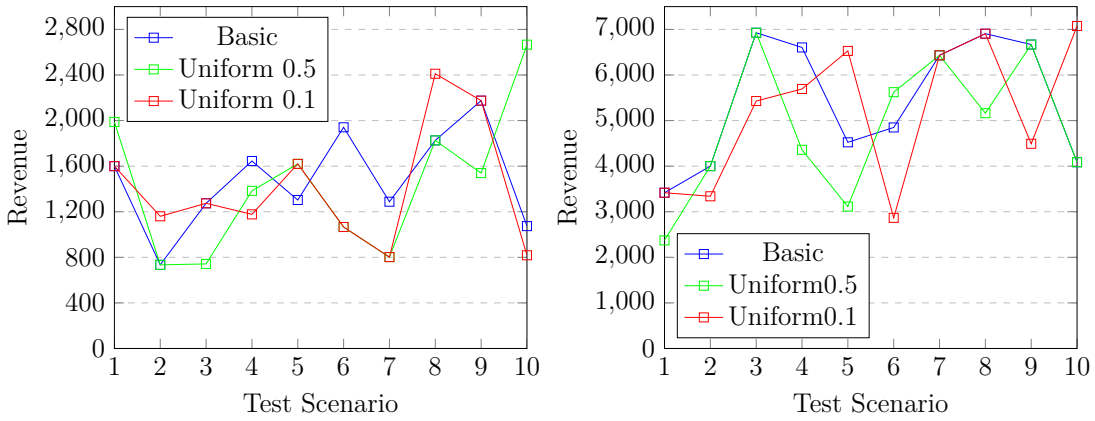


Table 7.1: Revenue Per Test Scenario of the Basic Variant and Variations of the Uniform Crossover Operator in the Small(left) and the Big(right) Environment

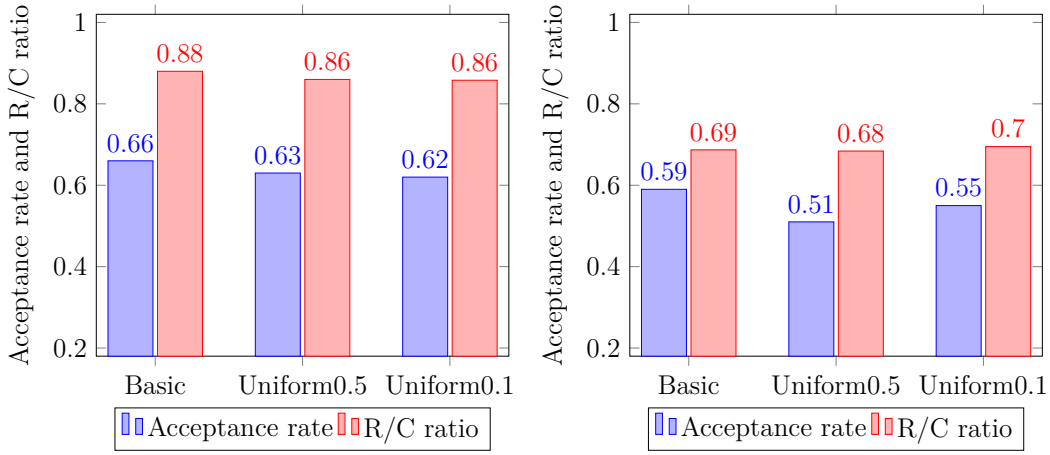


Table 7.2: Acceptance rate and R/C ratio for small(left) and big(right) environment

better than the others in terms of revenue. In the small test scenarios, each of the different operators had the best performance for one or more test scenarios. While Uniform 0.5 performed well in the first and last test scenario, this performance could not be confirmed in further test runs. The large discrepancy in the tenth test case between Uniform 0.5 and the other two operators appears to be an outlier result compared to the other test scenarios. Further tests have shown that 1-Point Crossover and Uniform 0.1 can also perform well in this scenario.
The situation is similar in the large test environment. The peaks and troughs of the respective graphs in Table 7.1(right) were exceptions and could not be maintained in repeated tests. A comparison of the acceptance rate and the R/C ratio between the different recombination operators is depicted in Table 7.2. There is little to no difference to the acceptance rate in all three variants for the small or the big test environment. The reduced R/C ratio and acceptance rate in the big test environment

Variant:	Basic	Uniform0.5	Uniform0.1
$\sum Revenue(S)$:	14860	14366	14103
ØRevenuePerTest(S)/SD:	1486/411	1437/594	1410/514
ØRevenuePerVNR(S)/SD:	225/89.6	228/94.7	227.5/96.3
ØSolving time(S):	34.32	31.7	32.3
Teriminations(S):	39	39	33
$\sum Revenue(B)$:	54413	48743	52175
ØRevenuePerTest(B)/SD:	5441.3/1318	4874.3/1466	5217.5/1503
ØRevenuePerVNR(B)/SD:	922.3/428.8	955.7/434	948.8/445
ØSolving time(B):	443.1	421.2	522.65
Terminations(B):	1	3	2

Table 7.3: Averages for the Basic Variant, Uniform0.5 and Uniform0.1 Crossover

stems from the fact that the nodes of VNs connect themselves with a probability of 0.7 while being generated, which results in considerably more links in VNs with increasing number of nodes within a VN, which makes it harder to embed and decreases the possibility of near optimal solutions. This last point can be seen in the rate of early terminations in Table 7.3, which is significantly lower than in the small testing environment. However, all variants did perform well in the small test environment. A R/C ratio of 0.88 shows a good utilization of substrate resources and two thirds of all requests were accepted without requiring the use of additional advanced mechanisms. Even in the big test environment all three variants performed equally acceptable with a R/C ratio of 0.7 and an acceptance rate above 50%, although most of the VNs in this scenario are relatively difficult to embed. It is important to note that this is only confirmed for the described test scenarios and does not ensure general good performance.
Table 7.3 shows a comparison between the operators regarding revenue, solving time and early terminations, as defined in chapter 6, for the small(S) and the big(B) test environments. Although Uniform0.1 performed worse in regards to average revenue per Test in the small environment with a value of 1410 in comparison to the basic variant with a value of 1486, the standard deviation of 411 and 514, respectively, shows that no significant difference could be found between the variants. This applies to all three recombination variations considered.
The results indicate that all three variants are possible candidates for further evaluations and can be considered as suitable operators due to the fact that none of them performed significantly worse than the others. However, 1-Point Crossover is used for the remaining evaluation tests for simplicity reasons.

Mutation Operators
Both mutation operators introduced in Section 6.1 are evaluated in this section. Applying the Inversion Mutation instead of the Simple Mutation, which is used in the

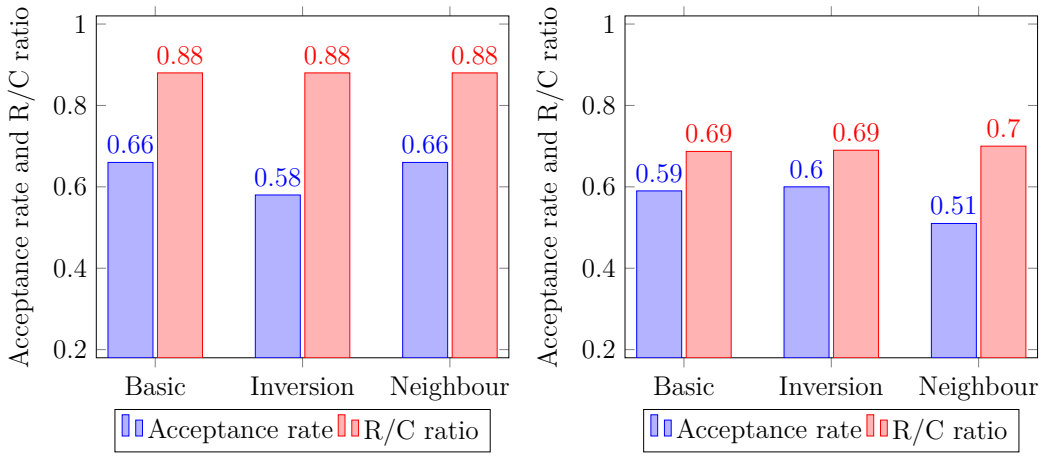


Table 7.4: Acceptance Rate and R/C Ratio for the Small(left) and Big(right) Environment

Variant:	Inversion	Neighbour
$\sum Revenue(S)$:	12882	14556
ØRevenuePerTest(S)/SD:	1288/491	1455.6/550.3
ØRevenuePerVNR(S)/SD:	222/95	220.5/87.8
ØSolving time(S):	33	30.4
Terminations(S):	34	39
$\sum Revenue(B)$:	56120	49449
ØRevenuePerTest(B)/SD:	5612/1683	4944.9/1800
ØRevenuePerVNR(B)/SD:	935.3/453.7	969.6/451
ØSolving time(B):	385	311.4
Terminations(B):	1	2

Table 7.5: Averages for Inversion and Neighbour Mutation

Basic GA, did not significantly affect the results regarding acceptance rate or R/C ratio in any of the two test environments. In comparison with the results of the basic variant and the results of the Neighbor Mutation depicted in Table 7.4 no significant improvements or deterioration could be observed. The acceptance rate of 0.58 in the small test environment could be improved with further test runs, as well as the overall revenue of 12882, as shown in Table 7.5. While the overall revenue of Inversion Mutation in the big test environment was highest so far, the standard deviation shows that this value is not statistically significant. However, Inversion Mutation has the potential to better utilize the resources of multiple nodes at once by reassembling the current node mapping solution and finding better suitable solutions regarding the substrate capacities, which could allow more embeddings in the long term. The test environment with the chosen bounds to capacities and demands could result in dete-

riorating the impact of this operator by not freeing enough capacities for subsequent embeddings, even if a profoundly more suitable embedding in terms of CPU resource utilization was found.
The Neighbour Mutation could not directly improve the investigated metrics in any of the test environments. In comparison with the Simple Mutation, as well as with the Inversion Mutation, no significant differences could be observed, which can be seen in Table 7.4 and Table 7.5. The relatively low value in acceptance rate of 0.51 in the big test environment could be improved by further tests, as well as the overall revenue shown in Table 7.5. This concept still has noteworthy potential by reducing the fragmentation regarding the location of virtual nodes within the substrate network. By favouring neighbouring nodes of a node mapping solution, it can lead to shorter communication channels and could therefore improve performance metrics which were not considered in this thesis. Just as with the recombination operators, it has to be stated that all three variants, the Simple Mutation, the Inversion Mutation and the Neighbour Mutation, did perform equally well. Even if no improvement could be directly derived, the results indicate that all variants can be considered in future investigations as possible candidates for a suitable mutation operator. However, the Simple Mutation scheme is used in subsequent evaluation tests for simplicity reasons.

Replacement
The evaluation results of the alternative Truncation Replacement strategy in comparison with the basic variant, namely Elitist Replacement, are presented in Table 7.6 and Table 7.7. Even though the acceptance rate was slightly higher in the small environment with 0.73 in comparison to the acceptance rate of 0.66 in one test run, this advantage could not be maintained through further tests and is therefore not representative. In general, the Truncation Replacement strategy does not have any significant impact in either of the test environments regarding the evaluation metrics, although it showed potential in single test scenarios, where it performed better than the basic variant, which resulted in slightly higher acceptance rates of 0.73 and 0.62 in the respective test environments. But the results of further test runs indicate that this cannot singly be attributed to the replacement strategy, due to the moderately high fluctuation of the performance while using Truncation Replacement.
In theory, Truncation Replacement has the potential to enhance the average fitness in each generation by having at least the predefined percentage consisting of high performing individuals as shown in Figure 6.4. However, this could not be verified by the obtained data due to varying performances of single chromosomes and varying numbers of feasible chromosomes in the final population.

Selection
Tournament Selection, described in Section 3.4 and chapter 6, is applied to choose chromosomes which undergo genetic alterations, i.e. crossover and mutation. The impact of this selection method is now presented.

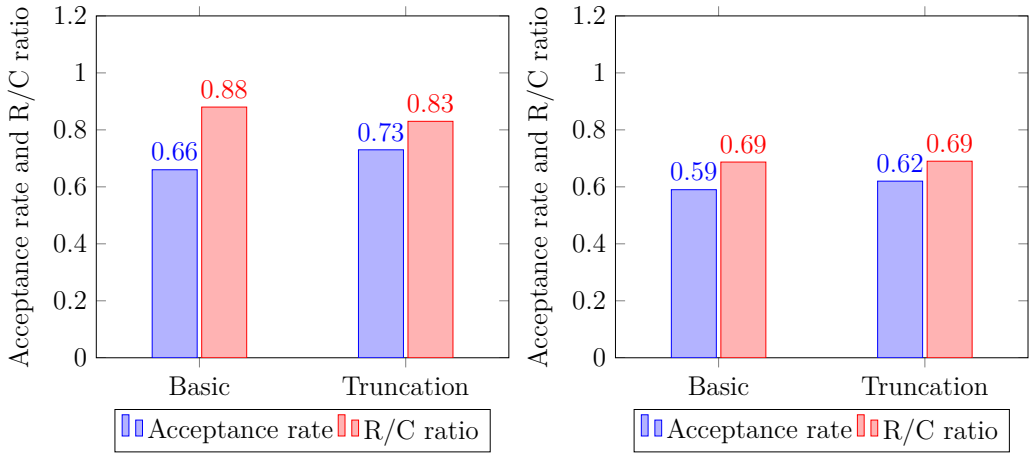


Table 7.6: Acceptance Rate and R/C Ratio for the Small(left) and Big(right) Environment

Variant:	Truncation
$\sum Revenue(S)$:	17191
ØRevenuePerTest(S)/SD:	1719.1/395.5
ØRevenuePerVNR(S)/SD:	235.5/89.7
ØSolving time(S):	30.25
Terminations(S):	39
$\sum Revenue(B)$:	58013
ØRevenuePerTest(B)/SD:	5801.3/1652
ØRevenuePerVNR(S)/SD:	935.7/519.8.7
ØSolving time(B):	422.6
Terminations(B):	1

Table 7.7: Averages for Truncation Replacement

As Table 7.8 shows, applying tournament selection did result in a noteworthy improvement in terms of acceptance rate in the small and even so in the big test environment and represents the highest values obtained so far. As a consequence, the overall revenue of 71962, shown in Table 7.9, is considerably higher than the 54413 generated by the basic variant. Considering the standard deviation of average revenue per test and average revenue per VNR as shown in Table 7.9, these values cannot be considered statistically significant. However, Tournament Selection did perform better than the basic variant in different previous test environments evaluated by the author, and showed good performance in continuous evaluations, indicating that Tournament Selection could indeed be an improvement for the Basic VNE GA. The performance of the basic version with random selection and the variant with Tournament Selection in the big test environment is shown in Table 7.10.

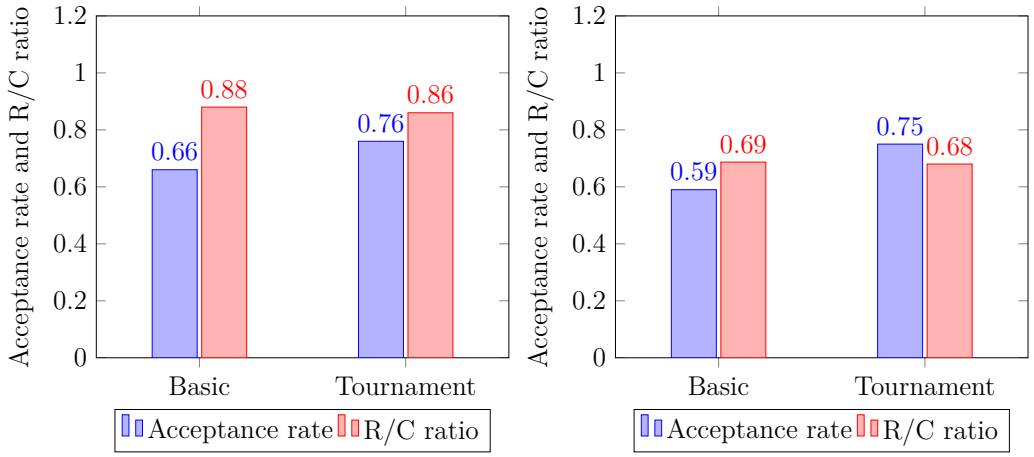


Table 7.8: Acceptance Rate and R/C Ratio for Small(left) and Big(right) Environment

Variant:	Tournament
$\sum Revenue(S)$:	17637
ØRevenuePerTest(S)/SD:	1763.7/458.4
ØRevenuePerVNR(S)/SD:	232.1/89.6
ØSolving time(S):	31.6
Terminations(S):	40
$\sum Revenue(B)$:	71962
ØRevenuePerTest(B)/SD:	7196.2/1894.7
ØRevenuePerVNR(B)/SD:	959.5/460.8
ØSolving time(B):	508.9
Terminations(B):	0

Table 7.9: Averages for Tournament Selection

As shown, Tournament Selection performs equally or better in every scenario in this example. Interestingly, Tournament Selection performed significantly better in the third scenario where it was able to embed nine out of ten VNs. However, further runs showed variable results, with an average acceptance rate of 0.66 in this specific test scenario. Regarding all evaluations so far, Tournament Selection still showed the most promising results.
To further investigate this promising operator, the small test environment is further utilized in the following way. Three out of ten test scenarios are run ten times each to obtain the average accepted VNs in those specific scenarios and make a more detailed comparison. The chosen scenarios had the highest discrepancy between the investigated implementations regarding the acceptance rate in the first test run. The scenarios are denoted as Scenario3 (S.3), Scenario5(S.5) and Scenario10(S.10). A fur-

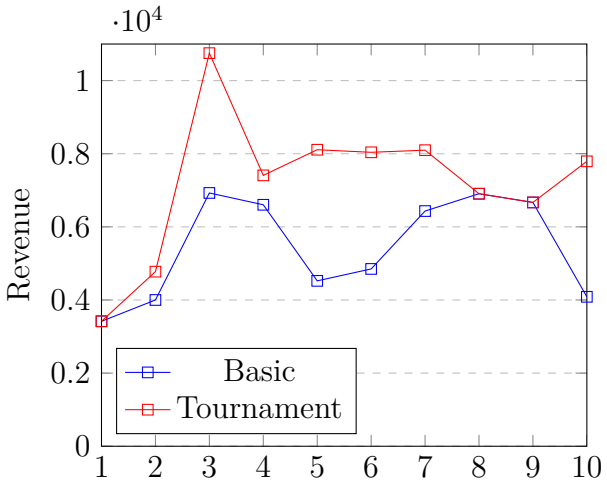


Table 7.10: Performance of Random Selection and Tournament Selection in the Scenario 10 of the Big Environment

ther investigation was carried out by also considering one big scenario denoted as Scenario10(Big) and performing ten runs with each implementation. The results can be seen in Table 7.11, while the chart in Table 7.12 exemplary shows the revenue generated by the S.10 test runs.

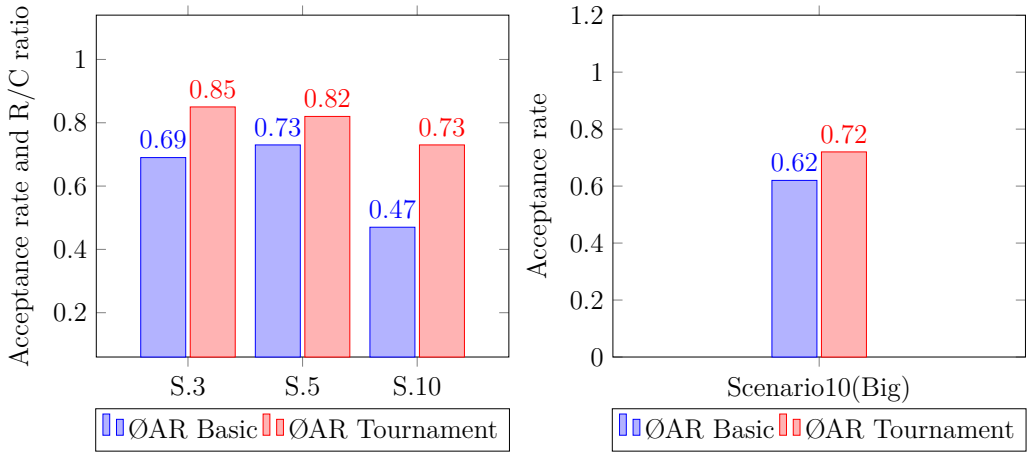


Table 7.11: Average Acceptance Rate for Small(left) and Big(right) Test Scenarios

These even more comprehensive tests show that the implementation equipped with Tournament Selection does perform better on average with regard to the acceptance rate and as a consequence the overall revenue. As can be seen in Table 7.12, Tournament Selection performs equally or better in nine out of ten test runs. This suggests that Tournament Selection is responsible for the performance improvement regarding

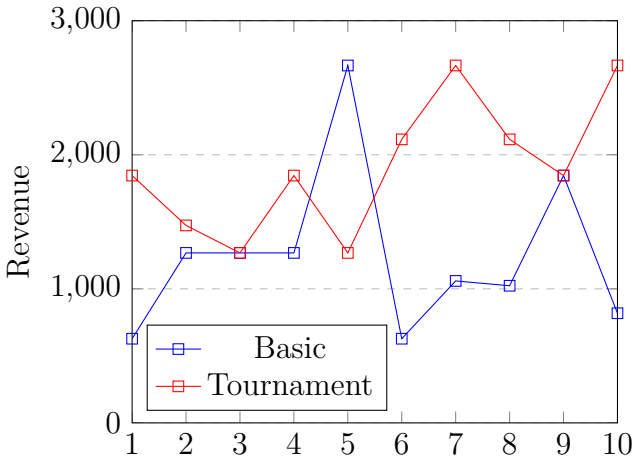


Table 7.12: Test Runs of Scenario 10 in the Small Environment

the acceptance rate, even though the standard deviation indicates no statistical significance.

Termination

The additional termination criteria defined in chapter 6 resulted in 263 early terminations in 700 runs in the small environment and 10 early terminations in the same number of runs in the big environment. The percentage of termination occurrences per generation is depicted in Table 7.13.

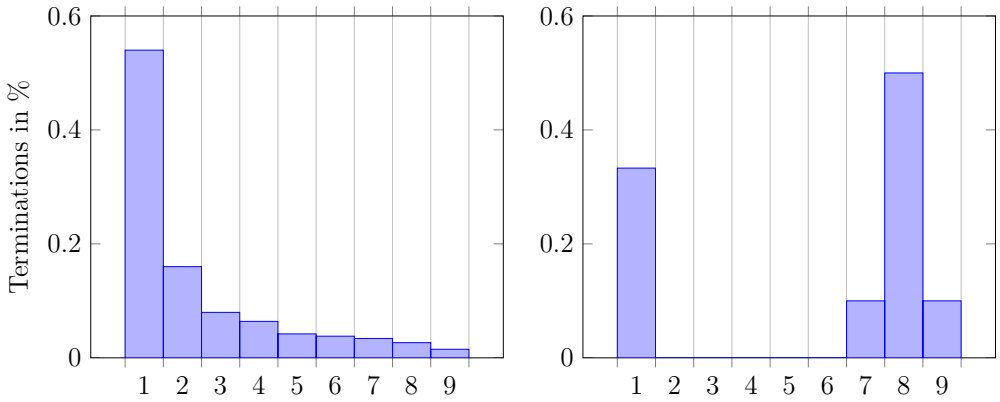


Table 7.13: Early Termination Occurrences per Generation for the Small(left) and the Big(right) Environment

Table 7.13 depicts the percentage of early termination occurrences *after* each generation. It shows that the most common occurrence of early termination is found after the first generation within the small environment. At this state, no genetic operators have been applied yet and the initialization procedure described in chapter 5 is

responsible for at least all terminations in generation one. It is probable that even more early terminations are due to the initialization method because that procedure is used again during the algorithm for the re-initialization of infeasible solutions. Early termination occurrences decrease rapidly in the small environment after the second generation and decrease further after every generation.
However, in the big test environment most early terminations occurred after the eighth generation, accounting for 50% of all early terminations. Nevertheless 33% of all terminations occurred after the first generation and therefore can be attributed to the initialization method. The time invested in more generations seems to pay off more in the big test environment regarding near optimal embedding solutions, assuming that the numbers of terminations are representative for the big test environment, this outcome is unclear, given only ten cases.
The initialization method does contribute a large percentage of early terminations in all test runs. It can be recommended that this mechanism deserves further investigation in the future, considering the results in Table 7.14.

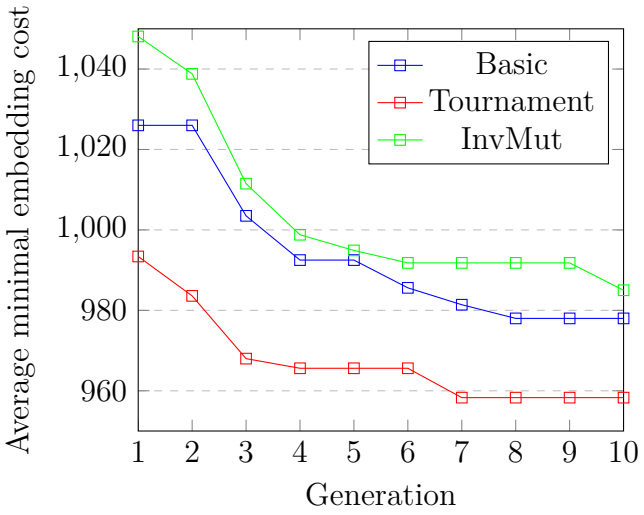


Table 7.14: Development of Minimal Embedding Cost

The figure in Table 7.14 shows the development of the average minimal embedding costs for each generation for three different implementations. We already discussed the high probability for the lack of significance between those different operators, however the depicted results all represent the same test scenario (scenario eight in the big environment) and the same number of accepted VNs in this scenario, which is ten out of ten. As expected, the average minimal embedding cost differs in the first generation due to the probabilistic nature of the initialization method. Interestingly, the quality of the first generation correlates with the overall solution quality in every subsequent generation for all three implementations - no overlaps can be found. This could indicate that the quality of the initialization determines the overall solution quality.
It is important to note that this is only suggestive and the course of the depicted graphs do not have to continue in this manner in subsequent generations, however it

could be an important indication.
One of the weaknesses of all presented variations is the long runtime. While this can be refined by more efficient programming, this shortcoming can also be overcome by parallelization and other concepts from the field of so-called *'Competent GAs'* [53].
Overall, the VNE GA presented performs better than expected regarding acceptance rate and R/C ratio. It has high potential for further optimizations and improvements considering the wealth of available information about improvement strategies for GAs. The high modularity of the algorithm allows for the implementation and testing of different strategies in regard to genetic operators, hybridization and parallelization, which can be seen as an extensive source of opportunities.

8 Summary and Future Work

This work introduced the VNE problem and its characteristics as well as the characteristics of its algorithms to the reader. A comprehensive introduction into GAs was given afterwards and related work regarding GAs in the context of VNE was presented. A possible basic implementation of a VNE GA was discussed in detail and further possible alterations of it's operators and extensions were considered and evaluated, in which the specific Tournament Selection operator turned out to be a possible improvement in regard to the acceptance rate of VNRs.

Many more techniques for efficiency enhancement of GAs are available and can be utilized to further improve the VNE GA. Parallelization, in the form of multiple populations is predestined to be used in GAs due to their already parallel nature and to efficiently cut down execution time [53, p.13]. The concept of *'Linkage Identification'* is an intriguing next research direction which enables optimal recombinations in GAs [50]. First steps towards hybridization in the form of integrated local search techniques were made by this author during the investigations documented in this thesis. These techniques are promising candidates for future improvements regarding the VNE GA due to their ability to safely develop solutions towards local optima which can be used for further genetic processing [53, p.13f]. An important direction for future research is the contemplation and integration of additional network characteristics like latency and storage into the VNE model to better represent real world applications. However this would make genetic operators probably even more disruptive and would consequently require more sophisticated repair mechanisms. Another evaluation platform than the mentioned ALEVIN project would be very beneficial as well. The ability to appropriate model real world networks and their defining characteristics in the context of new generation networks could greatly benefit and speed up the identification of important research directions.

Bibliography

[1] *ALEVIN2 download | SourceForge.net*. URL: https://sourceforge.net/projects/alevin/. (accessed: 30.03.2020).

[2] Lee Altenberg. "The Schema Theorem and Price's Theorem". In: *Foundations of Genetic Algorithms*. Vol. 3. Elsevier, 1995, pp. 23–49. ISBN: 978-1-55860-356-1. DOI: 10.1016/B978-1-55860-356-1.50006-6. URL: https://linkinghub.elsevier.com/retrieve/pii/B9781558603561500066 (visited on 02/22/2020).

[3] David G. Andersen. "Theoretical Approaches to Node Assignment". In: 2002.

[4] T. Anderson et al. "Overcoming the Internet impasse through virtualization". In: *Computer* 38.4 (Apr. 2005), pp. 34–41. ISSN: 0018-9162. DOI: 10.1109/MC.2005.136. URL: http://ieeexplore.ieee.org/document/1432642/ (visited on 03/05/2020).

[5] James E Baker. "Reducing bias and inefficiency in the selection algorithm". In: *Proceedings of the second international conference on genetic algorithms*. Vol. 206. 1987, pp. 14–21.

[6] Shumeet Baluja. "Population-Based Incremental Learning. A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning". In: (1994), p. 42.

[7] Andreas Blenk et al. "NeuroViNE: A Neural Preprocessor for Your Virtual Network Embedding Algorithm". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. IEEE INFOCOM 2018 - IEEE Conference on Computer Communications. Apr. 2018, pp. 405–413. DOI: 10.1109/INFOCOM.2018.8486263.

[8] Lashon B Booker et al. "Recombination". In: *Handbook of evolutionary computation* 97.1 (1997), p. C3.

[9] H J Bremermann. "Optimization Through Evolution and Recombination". In: (1962), p. 12.

[10] Anne Brindle. "Genetic algorithms for function optimization". In: (1980).

[11] Haotong Cao et al. "ER-VNE: A Joint Energy and Revenue Embedding Algorithm for Embedding Virtual Networks". In: *IEEE Access* 6 (2018), pp. 47815–47827. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2867562.

[12] Haotong Cao et al. "Exact solutions of VNE: A survey". In: *China Communications* 13.6 (June 2016), pp. 48–62. ISSN: 1673-5447. DOI: 10.1109/CC.2016.7513202.

[13] Haotong Cao et al. “Heuristic solutions of virtual network embedding: A survey”. In: *China Communications* 15.3 (Mar. 2018), pp. 186–219. ISSN: 1673-5447. DOI: 10.1109/CC.2018.8332001. URL: http://ieeexplore.ieee.org/document/8332001/ (visited on 01/04/2020).

[14] Jorge Carapinha and Javier Jiménez. “Network virtualization: a view from the bottom”. In: *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*. 2009, pp. 73–80.

[15] Jenna Carr. “An Introduction to Genetic Algorithms”. In: (2014), p. 40.

[16] Xiaolin Chang, X.M. Mi, and Jogesh Muppala. “Performance evaluation of artificial intelligence algorithms for virtual network embedding”. In: *Engineering Applications of Artificial Intelligence* 26 (Nov. 1, 2013), pp. 2540–2550. DOI: 10.1016/j.engappai.2013.07.007.

[17] Xiang Cheng et al. “Virtual network embedding through topology-aware node ranking”. In: *ACM SIGCOMM Computer Communication Review* 41.2 (Apr. 15, 2011), p. 38. ISSN: 01464833. DOI: 10.1145/1971162.1971168. URL: http://portal.acm.org/citation.cfm?doid=1971162.1971168 (visited on 01/03/2020).

[18] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. “ViNEYard: Virtual Network Embedding Algorithms with Coordinated Node and Link Mapping”. In: *IEEE/ACM Trans. Netw.* 20.1 (Feb. 2012), pp. 206–219. ISSN: 1063-6692. DOI: 10.1109/TNET.2011.2159308. URL: https://doi.org/10.1109/TNET.2011.2159308 (visited on 11/16/2019).

[19] N.M.M.K. Chowdhury and R. Boutaba. “Network virtualization: state of the art and research challenges”. In: *IEEE Communications Magazine* 47.7 (July 2009), pp. 20–26. ISSN: 0163-6804. DOI: 10.1109/MCOM.2009.5183468. URL: http://ieeexplore.ieee.org/document/5183468/ (visited on 11/13/2019).

[20] Lawrence Davis. “Adapting operator probabilities in genetic algorithms”. In: *Proceedings of the third international conference on Genetic algorithms*. 1989, pp. 61–69.

[21] Kenneth A. De Jong. “Genetic Algorithms Are NOT Function Optimizers”. In: *Foundations of Genetic Algorithms*. Vol. 2. Elsevier, 1993, pp. 5–17. ISBN: 978-0-08-094832-4. DOI: 10.1016/B978-0-08-094832-4.50006-4. URL: https://linkinghub.elsevier.com/retrieve/pii/B9780080948324500064 (visited on 02/19/2020).

[22] Kalyanmoy Deb. “Genetic Algorithm in Search and Optimization: The Technique and Applications”. In: (1998), p. 29.

[23] Nick Feamster, Lixin Gao, and Jennifer Rexford. “How to lease the internet in your spare time”. In: *ACM SIGCOMM Computer Communication Review* 37.1 (Jan. 22, 2007), p. 61. ISSN: 01464833. DOI: 10.1145/1198255.1198265. URL: http://portal.acm.org/citation.cfm?doid=1198255.1198265 (visited on 01/02/2020).

[24] Andreas Fischer et al. “Virtual Network Embedding: A Survey”. In: *IEEE Communications Surveys Tutorials* 15.4 (2013), pp. 1888–1906. ISSN: 1553-877X, 2373-745X. DOI: 10.1109/SURV.2013.013013.00155.

[25] David B Fogel. “A parallel processing approach to a multiple travelling salesman problem using evolutionary programming”. In: *Proceedings of the fourth annual symposium on parallel processing*. Fullerton, CA. 1990, pp. 318–326.

[26] David B. Fogel and Lauren C. Stayton. “On the effectiveness of crossover in simulated evolutionary optimization”. In: *Biosystems* 32.3 (Jan. 1994), pp. 171–182. ISSN: 03032647. DOI: 10.1016/0303-2647(94)90040-X. URL: https://linkinghub.elsevier.com/retrieve/pii/030326479490040X (visited on 02/19/2020).

[27] A.S. Fraser. “Simulation of Genetic Systems by Automatic Digital Computers”. In: *Australian Journal of Biological Sciences* 11.4 (1958), p. 603. ISSN: 0004-9417. DOI: 10.1071/BI9580603. URL: http://www.publish.csiro.au/?paper=BI9580603 (visited on 02/20/2020).

[28] David E Goldberg. “A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-Oriented Simulated Annealing”. In: (1990), p. 16.

[29] David E Goldberg. “Genetic algorithms in search”. In: *Optimization, and MachineLearning* (1989).

[30] David E. Goldberg and Kalyanmoy Deb. “A Comparative Analysis of Selection Schemes Used in Genetic Algorithms”. In: *Foundations of Genetic Algorithms*. Vol. 1. Elsevier, 1991, pp. 69–93. ISBN: 978-0-08-050684-5. DOI: 10.1016/B978-0-08-050684-5.50008-2. URL: https://linkinghub.elsevier.com/retrieve/pii/B9780080506845500082 (visited on 02/20/2020).

[31] Long Gong et al. “Toward profit-seeking virtual network embedding algorithm via global resource capacity”. In: *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. IEEE INFOCOM 2014 - IEEE Conference on Computer Communications. Apr. 2014, pp. 1–9. DOI: 10.1109/INFOCOM.2014.6847918.

[32] Randy L. Haupt and S. E. Haupt. *Practical genetic algorithms*. 2nd ed. Hoboken, N.J: John Wiley, 2004. 253 pp. ISBN: 978-0-471-45565-3.

[33] John Holland. “Adaptation in natural and artificial systems: an introductory analysis with application to biology”. In: *Control and artificial intelligence* (1975).

[34] Johannes Inführ and Günther Raidl. “A memetic algorithm for the virtual network mapping problem”. In: *Journal of Heuristics* 22.4 (2016), pp. 475–505.

[35] Manoj Kumar et al. “GENETIC ALGORITHM: REVIEW AND APPLICATION”. In: (2010), p. 4.

[36] Karl-Heinz Land. *Erde 5.0-die Zukunft provozieren*. FutureVisionPress eK, 2018.

[37] P Larran Larranaga et al. “Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators”. In: (1999), p. 42.

[38] G. E. Liepins and M. R. Hilliard. "Genetic algorithms: Foundations and applications". In: *Annals of Operations Research* 21.1 (Dec. 1989), pp. 31–57. ISSN: 0254-5330, 1572-9338. DOI: 10.1007/BF02022092. URL: http://link.springer.com/10.1007/BF02022092 (visited on 02/20/2020).

[39] Siew Mooi Lim et al. "Crossover and Mutation Operators of Genetic Algorithms". In: *International Journal of Machine Learning and Computing* 7.1 (Feb. 2017), pp. 9–12. ISSN: 20103700. DOI: 10.18178/ijmlc.2017.7.1.611. URL: http://www.ijmlc.org/index.php?m=content&c=index&a=show&catid=69&id=704 (visited on 02/21/2020).

[40] Jia Liu et al. "Research on virtual network mapping based on mixed genetic algorithm". In: *Journal of Chinese Computer Systems* 37.4 (2016), pp. 773–777.

[41] S. Marsili Libelli and P. Alba. "Adaptive mutation in genetic algorithms". In: *Soft Computing* 4.2 (July 2000), pp. 76–80. ISSN: 1432-7643. DOI: 10.1007/s005000000042. URL: http://link.springer.com/10.1007/s005000000042 (visited on 02/21/2020).

[42] S. Mashohor, J.R. Evans, and T. Arslan. "Elitist Selection Schemes for Genetic Algorithm based Printed Circuit Board Inspection System". In: *2005 IEEE Congress on Evolutionary Computation*. 2005 IEEE Congress on Evolutionary Computation. Vol. 2. Edinburgh, Scotland, UK: IEEE, 2005, pp. 974–978. ISBN: 978-0-7803-9363-9. DOI: 10.1109/CEC.2005.1554796. URL: http://ieeexplore.ieee.org/document/1554796/ (visited on 02/23/2020).

[43] Marcio Melo et al. "Optimal Virtual Network Embedding: Node-Link Formulation". In: *IEEE Transactions on Network and Service Management* 10.4 (Dec. 2013), pp. 356–368. ISSN: 2373-7379. DOI: 10.1109/TNSM.2013.092813.130397.

[44] Xiuming Mi et al. "Embedding Virtual Infrastructure Based on Genetic Algorithm". In: *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*. 2012 13th International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT). Beijing, China: IEEE, Dec. 2012, pp. 239–244. ISBN: 978-0-7695-4879-1. DOI: 10.1109/PDCAT.2012.71. URL: http://ieeexplore.ieee.org/document/6589270/ (visited on 02/25/2020).

[45] Gourav Misra et al. "Internet of Things (IoT) – A Technological Analysis and Survey on Vision, Concepts, Challenges, Innovation Directions, Technologies, and Applications (An Upcoming or Future Generation Computer Communication System Technology)". In: *American Journal of Electrical and Electronic Engineering* (2016), p. 10.

[46] Melanie Mitchell. "An Introduction to Genetic Algorithms". In: (1996), p. 162.

[47] Pablo Moscato. "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts Towards Memetic Algorithms". In: (1987), p. 69.

[48] Heinz Mühlenbein and Dirk Schlierkamp-Voosen. "Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization". In: *Evolutionary Computation* 1.1 (Mar. 1993), pp. 25–49. ISSN: 1063-6560, 1530-9304. DOI: 10.1162/evco.1993.1.1.25. URL: http://www.mitpressjournals.org/doi/10.1162/evco.1993.1.1.25 (visited on 02/20/2020).

[49] Heinz Mühlenbein and Hans-Michael Voigt. "Gene Pool Recombination in Genetic Algorithms". In: *Meta-Heuristics*. Ed. by Ibrahim H. Osman and James P. Kelly. Boston, MA: Springer US, 1996, pp. 53–62. ISBN: 978-1-4612-8587-8 978-1-4613-1361-8. DOI: 10.1007/978-1-4613-1361-8_4. URL: http://link.springer.com/10.1007/978-1-4613-1361-8_4 (visited on 02/23/2020).

[50] Masaharu Munetomo and David E Goldberg. "Identifying linkage by nonlinearity check". In: *IlliGAL Report* 98012 (1998).

[51] Khoa T.D. Nguyen and Changcheng Huang. "An Intelligent Parallel Algorithm for Online Virtual Network Embedding". In: *2019 International Conference on Computer, Information and Telecommunication Systems (CITS)*. 2019 International Conference on Computer, Information and Telecommunication Systems (CITS). Aug. 2019, pp. 1–5. DOI: 10.1109/CITS.2019.8862072.

[52] Norbert Niebert et al. "Network Virtualization: A Viable Path Towards the Future Internet". In: *Wireless Personal Communications* 45.4 (June 1, 2008), pp. 511–520. ISSN: 1572-834X. DOI: 10.1007/s11277-008-9481-6. URL: https://doi.org/10.1007/s11277-008-9481-6 (visited on 01/02/2020).

[53] Kumara Sastry, David E. Goldberg, and Graham Kendall. "Genetic Algorithms". In: *Search Methodologies*. Ed. by Edmund K. Burke and Graham Kendall. Boston, MA: Springer US, 2014, pp. 93–117. ISBN: 978-1-4614-6939-1 978-1-4614-6940-7. DOI: 10.1007/978-1-4614-6940-7_4. URL: http://link.springer.com/10.1007/978-1-4614-6940-7_4 (visited on 01/21/2020).

[54] William M Spears and Kenneth D De Jong. *On the virtues of parameterized uniform crossover*. Tech. rep. Naval Research Lab Washington DC, 1995.

[55] M. Srinivas and L.M. Patnaik. "Genetic algorithms: a survey". In: *Computer* 27.6 (June 1994), pp. 17–26. ISSN: 0018-9162. DOI: 10.1109/2.294849. URL: http://ieeexplore.ieee.org/document/294849/ (visited on 02/23/2020).

[56] Gilbert Syswerda. "Uniform crossover in genetic algorithms". In: *Proceedings of the 3rd international conference on genetic algorithms*. 1989, pp. 2–9.

[57] Shigeyoshi Tsutsui, Masayuki Yamamura, and Takahide Higuchi. "Multi-parent Recombination with Simplex Crossover in Real Coded Genetic Algorithms". In: (1999), p. 8.

[58] J.S. Turner and D.E. Taylor. "Diversifying the Internet". In: *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005*. GLOBECOM '05. IEEE Global Telecommunications Conference, 2005. Vol. 2. ISSN: 1930-529X. Nov. 2005, 6 pp.–760. DOI: 10.1109/GLOCOM.2005.1577741.

[59] K Tutschku et al. “Network Virtualization: Implementation Steps Towards the Future Internet”. In: 17 (2009), p. 15.

[60] Umbarkar and Sheth. “CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW”. In: *ICTACT Journal on Soft Computing* 06.1 (Oct. 1, 2015), pp. 1083–1092. ISSN: 09766561, 22296956. DOI: 10.21917/ijsc.2015.0150. URL: http://ictactjournals.in/ArticleDetails.aspx?id=2109 (visited on 02/21/2020).

[61] B.M. Waxman. “Routing of multipoint connections”. In: *IEEE Journal on Selected Areas in Communications* 6.9 (Dec. 1988), pp. 1617–1622. ISSN: 07338716. DOI: 10.1109/49.12889. URL: http://ieeexplore.ieee.org/document/12889/ (visited on 03/24/2020).

[62] Saneh Lata Yadav and Asha Sohal. “Comparative Study of Different Selection Techniques in Genetic Algorithm”. In: *International Journal of Engineering* 6.3 (2017), p. 7.

[63] Jin Y Yen. “Finding the k shortest loopless paths in a network”. In: *management Science* 17.11 (1971), pp. 712–716.

[64] Minlan Yu et al. “Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration”. In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), p. 11.

[65] Hongnian Zang, Shujun Zhang, and Kevin Hapeshi. “A Review of Nature-Inspired Algorithms”. In: *Journal of Bionic Engineering* 7 (S4 Dec. 2010), S232–S237. ISSN: 1672-6529, 2543-2141. DOI: 10.1016/S1672-6529(09)60240-7. URL: http://link.springer.com/10.1016/S1672-6529(09)60240-7 (visited on 02/23/2020).

[66] Peiying Zhang, Haipeng Yao, and Yunjie Liu. “Virtual Network Embedding Based on the Degree and Clustering Coefficient Information”. In: *IEEE Access* 4 (2016), pp. 8572–8580. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2632421.

[67] Sheng Zhang, Jie Wu, and Sanglu Lu. “Virtual network embedding with substrate support for parallelization”. In: *2012 IEEE Global Communications Conference (GLOBECOM)*. 2012 IEEE Global Communications Conference (GLOBECOM). ISSN: 1930-529X. Dec. 2012, pp. 2615–2620. DOI: 10.1109/GLOCOM.2012.6503511.

[68] Y. Zhu and M. Ammar. “Algorithms for Assigning Substrate Network Resources to Virtual Network Components”. In: *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*. Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications. Apr. 2006, pp. 1–12. DOI: 10.1109/INFOCOM.2006.322.