VIRUS TRANSMISSION GENETIC ALGORITHM

by

WEIXIN LING

(Under the Direction of Walter D. Potter)

ABSTRACT

In this thesis a new Virus Transmission Genetic Algorithm is presented based on the modern progress of biological evolution. The VTGA simulates the evolution of immune defense and the infection transmission model. Simulating biological infections, this algorithm contains one virus population and one host population as well as several new operations, including virus infection, virus spread and virus evolution. To study the effectiveness of the VTGA, we apply it to several function optimization problems, several travelling salesman problems and two forest planning problems. We discuss in this thesis how the VTGA's performance reacts to different configurations of parameters. Results of experiments show that the VTGA performs well at searching optimal solutions and preserving diversity of population.

INDEX WORDS: Virus Transmission Genetic Algorithm, Forest Planning Problem

VIRUS TRANSMISSION GENETIC ALGORITHM

by

WEIXIN LING

B.Eng., South China Agricultural University, China, 2006

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment

of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2012

VIRUS TRANSMISSION GENETIC ALGORITHM

by

WEIXIN LING

Major Professor:	Walter D. Potter
Committee:	Khaled Rasheed
	Pete Bettinger

Electronic Version Approved:

Maureen Grasso

Dean of the Graduate School

The University of Georgia

December 2012

DEDICATION

To my grandmother, who passed away in China while I was pursuing my degree. Since she raised me when I was a child, I want to express my greatly appreciation to her.

ACKNOWLEDGEMENTS

I would like to thank my major professor Dr. Potter for his instruction during my study at the Institute for Artificial Intelligence at the University of Georgia. And I also want to thank my committee members: Dr. Rasheed and Dr. Bettinger for their cooperation during my thesis work. Finally, I would like to thank my mother for her understanding and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The Genetic Algorithm is a very popular population-based search and optimization method [1] [2] [3]. Despite its great success, the GA has a problem called premature convergence. The reason causing this problem is because a few outstanding solutions dominate the population. Therefore, the lack of diversity may trap the algorithm in a local optimum. Solving this problem without changing the GA is impossible because the feature of the GA determines that the algorithm needs to converge towards some points during the search. Commonly, to make sure that the GA gives us the global best solution, we run the GA many times, hoping it converges to different solutions each time and one of them is the global optimum. The Island GA follows this idea but was developed to be more efficient with parallel computation. And there are other differences between the Island GA and running the simple GA many times. Firstly, the Island GA allows multiple GA populations to evolve simultaneously using parallel computation instead of running several trials of the GA serially. Secondly, during the computation, the Island GA selects a few outstanding individuals from one population and sends them to the next population on a ring or a torus. However, neither running many trials of the GA nor running the Island GA guarantees the algorithm will converge to the global optimum. More details of the simple GA and the Island GA will be discussed in chapter 2.

Another method for solving the premature convergence problem is storing patterns during a search. Hence, effective patterns won't be eliminated from the search space when the algorithm converges. In that case, the diversity will be preserved. The Virus-Evolutionary

Genetic Algorithm is an algorithm using this method. There are two populations of the VEGA. One is the host population, which works basically the same as the population of the GA. The other is the virus population. The virus population is for saving effective patterns during a search. When a new generation of the host population is created, these effective patterns offer information to make the new hosts better than their parents. This sounds to be a reasonable method for improving the GA. However, there are problems with the VEGA. The first problem is that, in the VEGA, every virus represents a continuous chromosome region, not a complete solution. If the effective patterns of a problem don't exist as continuous regions, then the virus population can't store much useful information. The second problem is how to determine the fitness of the viruses. Since the virus population of the VEGA has a limited size, apparently not all of the patterns found by the algorithm will be stored. In such a case, only the most effective patterns will be stored and we need to evaluate patterns to decide which patterns stay. In the VEGA, the evaluation of a virus involves many evaluations of the hosts related to the virus. So the evaluation of viruses is very complex. We will have further discussion about the VEGA in the Chapter 2.

Considering the reasonableness of storing effective patterns and the problem of the VEGA, we propose a new search algorithm called the Virus Transmission Genetic Algorithm. The VTGA uses two populations like the VEGA, one virus population and one host population. However, the populations work differently from that of the VEGA. The virus population of the VTGA is used for search and the host population of the VTGA is used for storing good schemes. In the VTGA, we simulate the evolution of viruses inside their hosts. Therefore, we have crossover and mutation operators implemented in the VTGA and they work similarly as that of the simple GA. Besides, in order to mimic the behaviors of viruses, we also implement infection,

spread and recovery operators following the virus transmission model. We will introduce the VTGA more in the Chapter 3.

To examine the efficiency of the VTGA, we implement programs of the algorithm for solving some function optimization problems, some travelling salesman problems and two forest planning problems. In the process of solving one of the function optimization problems, the Rastrigin function, we will discuss how to optimize the parameter values of the VTGA and how the change of the parameter values that affect the performance of the algorithm. After we have the best configuration, we compare the result of the Rastrigin function acquired by the VTGA with that acquired by the VEGA. We will also present results for other function optimization problems in the same section as well. In the second group of experiments, we use the VTGA to solve several well-known travelling salesman problems. And last but not least, we use the VTGA to solve two forest planning problems in the third group of experiments. Forest planning problems are considered to be complicated constrained optimization problems. In this paper, we will solve one 73-stand and one 625-stand forest planning problems. Discussion about the experiments is in the Chapter 4.

CHAPTER 2

BACKGROUND

2.1 GENETIC ALGORITHM

The genetic algorithm (GA) is a successful heuristic search method developed by John Holland in 1975 [1]. The GA mimics the natural selection process in Darwin's theory. It starts with a large population of stochastic solutions. The key idea in the GA is keeping good solutions in its population longer so that they have more opportunities to reproduce better offspring by combining or adjusting their chromosomes. A standard GA is composed of operators like selection, recombination and mutation. We are going to briefly discuss these operators one by one.

Selection operators are for picking parent individuals for reproduction. Good solutions with higher fitness values are more likely to be selected. Selected solutions have more opportunity to perform mating or stay in the next generation. There are many alternative selection methods for different problems. Fitness proportional selection (FPS) and tournament selection (TS) are two common methods of selection implementation. FPS computes a probability for every individual of how likely it will be selected. The probability is related to the absolute fitness value of an individual compared to the sum of fitness values of the whole population. But there are two known problems with FPS. One is premature convergence caused by dominance of a few outstanding individuals. The other one is lack of selection pressure. When the difference between fitness values is not obvious, the GA selects parents less heuristically but more randomly. Another commonly used selection strategy is tournament

selection (TS). Unlike FPS, tournament selection doesn't require information of the whole population. Every selection picks only a few candidates from the population, among which the one with the highest fitness value will be selected.

Recombination is another important feature of the genetic algorithm. It ensures that new solutions generated by the GA combine partial chromosomes from two or more "parent" solutions, inheriting effective patterns. Common recombination is generally one point or N-point crossover. One point crossover interchanges two parents at one randomly picked position to form two offspring chromosomes. In N-point crossover two or more positions are selected for interchanging. If children chromosomes are not able to inherit useful patterns with one point or N-point crossover due to features of problems, uniform crossover is a good alternative when good schemas are not formed by consecutive genes. In uniform crossover, every position of a child individual is considered independently whether the GA exchanges their values or not. And there are others. For some complex problems, in which we use uncommon chromosome encoding schemes, specialized crossover operators have been studied, for example, order crossover, edge recombination, and partially matched crossover [2] [3].

While recombination links up partial sequences that already exist in the population, mutation creates new solutions by breaking existing patterns and introducing new values to genes. In biology, mutation is usually harmful. There are many known human diseases caused by genetic mutation, also known as genetic disorder in medicine. However, in the genetic algorithm, mutation helps form new schemata so that the population has an opportunity to escape when trapped in a local optimum. Helpful as mutation is, we don't want too much mutation in the GA, because it is difficult for the GA to accumulate good patterns otherwise. So it is effective to maintain a low probability of mutation without damaging inheritance too much. Although the

most common way to implement mutation is bit-flipping, like recombination, there are many other schemes for mutation that have been proposed.

In the Virus Transmission Genetic Algorithm, we use similar crossover and mutation operators as that of the GA. That's why we consider that the VTGA is a new version of the GA instead of a completely new search algorithm.

2.2 ISLAND GENETIC ALGORITHM

Since parallel computation became attractive in 1980, people have been considering using this technique to improve evolutionary algorithms [4] [5], one type of which is the GA. The parallel implementation of the Genetic Algorithm, or Island GA, has several populations running the process of GA evolution in different processors or even different computer nodes. In order to make multiple populations cooperate as one program instead of several independent GAs, the parallel GA is commonly working within a communication structure, which is usually a ring or a torus. A communication structure performs migrations among populations every several generations. In the process of a migration, a few outstanding individuals are selected from every population. These individuals then will be sent to the neighboring population on the structure and replace the worst individuals of the neighboring populations.

To develop good heuristic search algorithms, the genetic algorithm or any others requires a trade-off to balance exploration of unexplored regions of the search space and exploitation around known good solutions. A simple genetic algorithm, because of possible dominance in the population, is likely to convergence towards a local optimum. When that happens, the GA explores only the region around the local optimum. But in the Island GA, because sub-GAs are running simultaneously, Martin et al suggest that when populations evolve independently around

the good solutions they contain [6], the Island GA actually explores different regions of the search space at the same time. After several generations, good solutions of each of those regions are developed. With communication, these good solutions of various search regions are placed together possibly with radically different patterns. Therefore, it is possible for offspring to inherit different schemata from diverse parents.

In our study of the VTGA, we want to utilize the advantage of isolated subpopulations of the Island GA. Following the idea, we keep every host individual isolated in the VTGA. So we just need one host population for preserving diversity instead of many in the algorithm. Hopefully by this method, we could have the advantage of the Island GA.

2.3 VIRUS-EVOLUTIONARY GENETIC ALGORITHM

With the recent progress of molecular biology, new theories of evolution, such as the neutral theory of molecular evolution and the virus theory of evolution, have been proposed [7] to supplement Darwin's theory of natural selection. Inspired by the virus theory of evolution, Kubota et al. proposed a new algorithm named the Virus-Evolutionary Genetic Algorithm (VEGA) [8] [9] (Figure 1, Figure 2, Figure 3). VEGA contains two populations: a host population and a virus population. The host population of VEGA performs identically as a regular population in a normal genetic algorithm. Meanwhile, the most important idea in VEGA is that the virus population has two virus infection operators, the reverse transcription operator and the transduction operator. In biology, reverse transcription is the process of viruses using the enzyme to reverse-transcribe their RNA genomes into DNA, which is then integrated into the host genome and replicated along with it. In VEGA, the reverse transcription operator performs similarly. Every several generations, a few virus individuals have a chance to transcribe their

substrings on the chromosome of some host individuals. And the transduction operator is used for creating or modifying virus individuals. Because in VEGA, what we want to preserve is useful patterns of solutions, so we use the transduction operator to extract substrings from host individuals and let those substrings be the chromosomes of viruses.

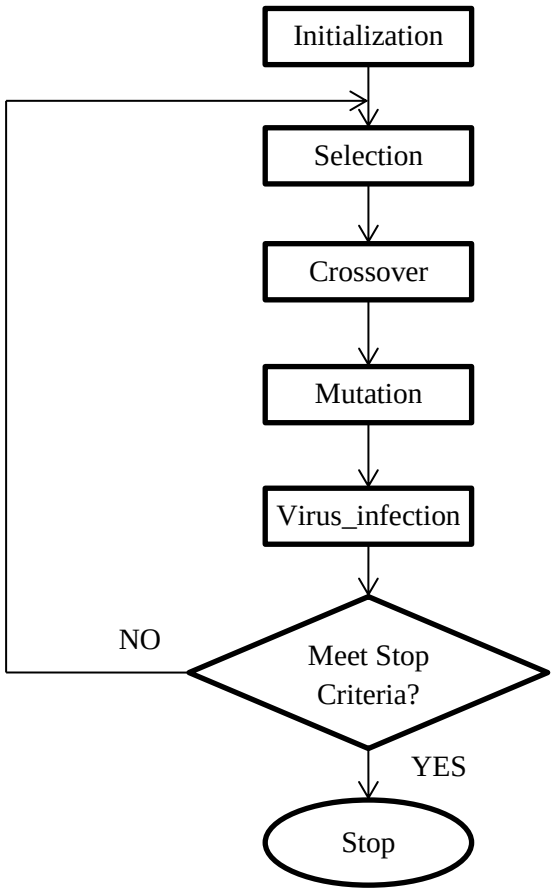


Figure 1: Flow Chart for Virus-Evolutionary Genetic Algorithm

Suppose there is a virus indicated by *i*. We assume that $fithost_j$ and $fithost'_j$ are the fitness values of a host individual *j* before and after the being infected by virus *i*, respectively. $fitvirus_{i,j}$ indicates how much virus *i* can improve host *j*. It denotes the difference between

$fithost_j$and $fithost'_j$. $fitvirus_i$ is the sum of fitness improvement of all host infected by virus *i*. Equations are listed below,

$$fitvirus_{i,j} = fithost'_j - fithost_j$$

$$fitvirus_i = \sum_{j \in S} fitvirus_{i,j}$$

where *i* is the virus number and *S* is a set of host individuals infected by the virus *i*. The fitness value of a virus indicates the overall influence from the virus to every host it infects. When the value is positive, that means the virus has positive effect on its host. Otherwise, if the value is negative, that means the virus doesn't improve its hosts much. And the fitness of a virus also controls how a virus survives. A so-called life force variable is computed by the following equation:

$$life_{i,t+1} = r \times life_{i,t} + fitvirus_i$$

where *t* and *r* mean the generation and the so-called life reduction rate, respectively. The procedure of a virus infection is shown in Figure 2. In every generation, the fitness value of every virus is computed. If the $life_{i,t}$ is negative, that means the virus doesn't have any positive influence on its hosts anymore, so the transduction operator rewrites the virus's chromosome by extracting a random substring from a randomly selected host. If the $life_{i,t}$ is positive, the virus individual obtains a partial substring from one of the hosts it infects.

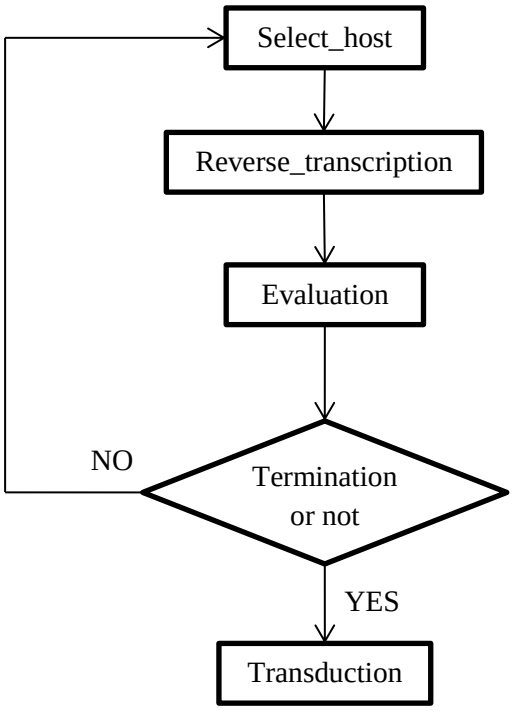


Figure 2: Flow Chart of Virus Infection

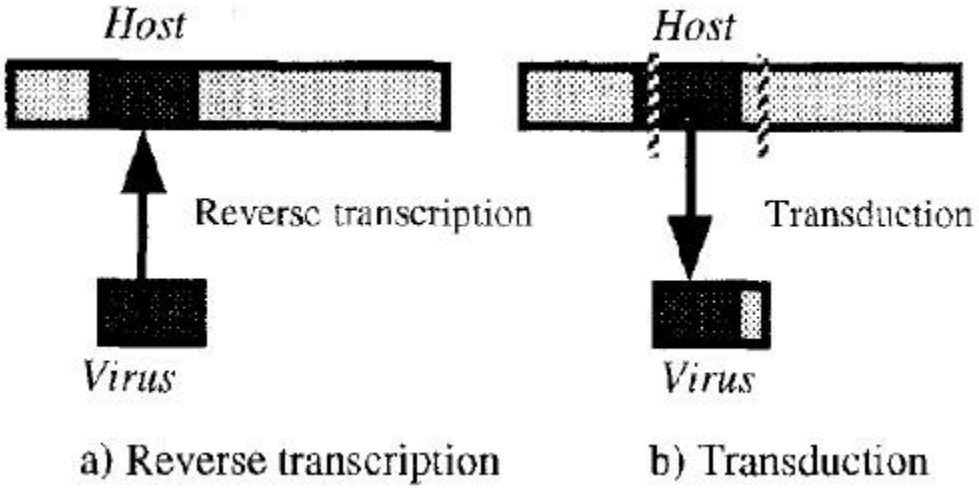


Figure 3: Virus Infection Operators

In the VEGA, the virus population is used for recording patterns and the host population is used for evolution. In the VTGA, we follow the configuration of two types of populations. So the algorithm has one virus population and one host population as well. However, we use the virus population for evolution and use the host population for recording good patterns.

2.4 ARTIFICIAL IMMUNE SYSTEM AND HEURISTIC SEARCH WITH IMMUNITY

Inspired by the virus transmission model, the key idea of the VTGA involves several biological facts of the natural immune systems. In computer science, there is a field concerned with the natural immune systems as well, which is called the Artificial Immune System (AIS). However, the VTGA is an algorithm very different from the AIS. The field of the AIS studies how to abstract the structure and function of the immune system to computational systems. In biology, the main roles of the immune system are to perform pattern recognition and to eliminate nonself or malfunctioning cells. Therefore, the field of the AIS focuses on solving computational problems with pattern recognition techniques [10].

On the other hand, in our study, although the word “immune system” is mentioned several times, we don’t actually try to implement any functions of the biological immune system here. We focus more on the role of the immune system in the coevolutionary relationship between hosts and viruses. That means when there are stronger viruses there will be stronger hosts with better immune systems, because those weak ones are killed by the viruses. In the VTGA, we simplify the immune system as a filter that preventing bad solutions being saved by the host population.

Nevertheless it will be interesting to think about combining the AIS with the VTGA considering the concerns they share in the immune system. But that is beyond the scope of this thesis, so we will save this topic for future studies.

2.5 EVOLUTION OF IMMUNE DEFENSES AND RESISTANCE TRAITS

Infectious diseases have been playing a key role in human evolutionary history. The interaction between human beings and pathogens is complicated. Every change in human behaviors, for instance animal domestication, urbanization and migration has great impact, positive or negative, on infection transmission. On the other hand, epidemics affect the evolution of people all the time. During 1348 – 1349 [11], the Black Death killed about one third of all Europeans. During 1918 the influenza pandemic killed 20 – 40 million people worldwide [12]. Moderate disease-induced mortality or lost opportunities for reproduction is likely to generate evolutionary responses over longer time scales. That means when epidemics hit human society weak individuals are likely to be eliminated by the diseases. Only individuals, who are strong enough to fight the diseases, have opportunities to survive. Therefore, in the future, offspring of the survivors would all be immune to those diseases.

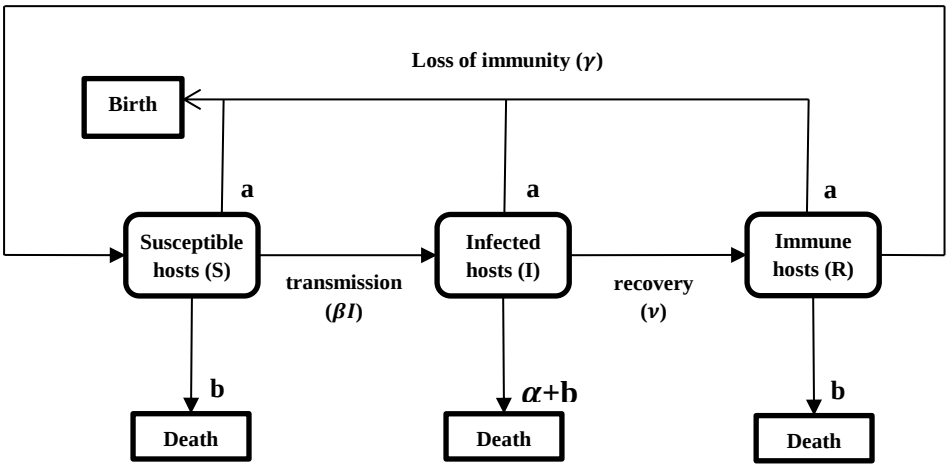
This process can also be explained as parasite-mediated selection. This selection increases the frequency of resistant phenotypes in a population. When an infection happens, those people who have a resistant phenotype are more likely to survive. One best-known example [13] is hemoglobin-related disorders that confer resistance to Plasmodium, including sickle-cell anemia and glucose 6-phosphate dehydrogenase deficiency. Balancing selection on both of these traits arises is a trade-off between malaria resistance and severe anemia [14].

Another example is the gene deletion CCR5-Δ32, an allele that confers resistance to HIV infection. This allele is only found in European populations. But it's absent from the native population of Africa, which is considered to be the origin of the HIV virus. That means this HIV-resistant gene is not caused by HIV. With study on molecular genetic data, people now believe that what causes this genetic difference is possibly the plague in Europe between 1300 and 1600 [15] or the smallpox epidemic in European populations [16]. In general, this is what happens. Before the plague or the smallpox happened, the population of people who have gene deletion CCR5-Δ32 is really small and negligible, because otherwise we would have found the same disorder in Africa. When the epidemics hit Europe, people with this genetic disorder are likely to survive. So these people are selected by infectious diseases. In our VTGA, we want to simulate this process. Every host population indicates one people. If a virus infecting this host has a better fitness than that of the host, the virus kills the host. As a result a new host is generated carrying the chromosome of the virus, which indicates that a stronger new host replaces the previous host. Another useful idea from this example is that geographical isolation helps with preserving diversity of genetic information. That's why CCR5-Δ32 is never found in Africa. If we want to develop a variation of the genetic algorithm using virus evolution, this is also beneficial to be included in our design. Therefore, in the VTGA, information won't be exchanged among host individuals directly by operators like crossover.

2.6 VIRUS TRANSMISSION MODEL

Unlike the VEGA, which simulates only the behavior of virus infection, the VTGA mimics the coevolution between virus and host populations. One important thing that happens in coevolution is the transmission of viruses. It's difficult to explain the process of virus transmission in detail.

The mechanism of real-world infection is not always obvious. But in modern epidemiology, the utility of mathematical models gives us the ability to describe infectious disease behavior with functions and parameters. One basic model of epidemics is called the Susceptible-Infectious-Recovered model (SIR) [17]. In the SIR model (Figure 4), individuals are classified into three statuses or three compartments as we show in Figure 4. Suppose we have one host population with a size N. The sum of S, I and R equals to N. Being susceptible means a host could be infected by viruses. Individuals are removed from the susceptible class either by natural mortality (b) or by infection after efficient contact with an infected host (at rate βSI). Infected hosts are removed by natural mortality (b), diseased-induced mortality (α) or recovery (ν) to the recovered class (γ). In our algorithm, we simulate the transmission of viruses following the SIR model. We use the infection and the recovery operators to mimic a virus's going in to a host and coming out from a host respectively. A host is always susceptible because in the VTGA, it can be infected anytime. If a host is infected by any viruses, it is infectious, meaning there are viruses inside it and they are able to come out to infect other hosts. Last but not least, if a host is improved so that no virus inside the host is better than it, gradually all viruses will leave that host to infect a weaker host. Once a host clears all viruses out of it, it is recovered.



$$\frac{dS}{dt} = a(S + I + R) - bS - \beta SI + \gamma R$$

$$\frac{dI}{dt} = \beta SI - (\alpha + b + \nu)I$$

$$\frac{dR}{dt} = \nu I - (b + \gamma)R$$

$$\frac{dN}{dt} = (a - b)N - \alpha I$$

Figure 4: Schematic Diagram and Differential Equations for a Typical SIR Compartment Model for a Directly Transmitted Microparasite.

2.7 FUNCTION OPTIMIZATION

In order to examine the searching ability of the algorithm, at the first stage of experiments we have some quality tests using several famous test functions [18].

De Jong's function

$$f(x) = \sum_{i=1}^{n} x_i^2$$

Test area is restricted to hypercube $-5.12 \leq x_i \leq 5.12, i = 1, \dots, n$. Global minimum$f(x) = 0$ is obtainable for $x_i = 0, i = 1, \dots, n$.

Axis parallel hyper-ellipsoid function

$$f(x) = \sum_{i=1}^{n} (i \cdot x_i^2)$$

Test area is restricted to hypercube $-5.12 \leq x_i \leq 5.12, i = 1, \dots, n$. Global minimum equal $f(x) = 0$ is obtainable for $x_i = 0, i = 1, \dots, n$.

Rotated hyper-ellipsoid function

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{i} x_j^2$$

Test area is restricted to hypercube $-65.536 \leq x_i \leq 65.536, i = 1, \dots, n$. Its global minimum equal $f(x) = 0$ is obtainable for $x_i = 0, i = 1, \dots, n$.

Rosenbrock's valley

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

Test area is restricted to hypercube $-2.048 \leq x_i \leq 2.048, i = 1, \dots, n$. Its global minimum equal $f(x) = 0$ is obtainable for $x_i, i = 1, \dots, n$.

Rastrigin's function

$$f(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)]$$

Test area is restricted to hypercube $-5.12 \leq x_i \leq 5.12, i = 1, \dots, n$. Its global minimum equal $f(x) = 0$ is obtainable for $x_i = 0, i = 1, \dots, n$.

Schwefel's function

$$f(x) = \sum_{i=1}^{n} [-x_i \sin(\sqrt{|x_i|})]$$

Test area is restricted to hypercube $-500 \leq x_i \leq 500, i = 1, \dots, n$. Its global minimum $f(x) = -418.9829n$ is obtainable for $x_i = 420.9687, i = 1, \dots, n$.

2.8 TRAVELLING SALESMAN PROBLEM

The travelling salesman problem (TSP) is a well-known NP-hard problem in combinatorial optimization. Given a list of cities and the distance between every pair of them, the TSP objective is to find the shortest route for a travelling salesman who has to visit every city on that list and return to where he starts at the beginning. The main difficulty of this problem is the immense number of possible tours: (n-1)!/2 for a list sized n. Suppose we have an $n \times n$ matrix $D = (d_{i,j})$ where $d_{i,j}$ is a nonnegative number representing the distance between city i and city j. Then the objective of the TSP problem is to find a permutation π from 1 to n minimizing:

$$\sum_{i=1}^{n} d_{i,\pi(i)} + d_{n,\pi(1)}$$

In this thesis, we will solve several travelling salesman problems with our algorithm. Because of the nature of the TSP problems, the representation is different from the representation for the function optimization problem, therefore, we will implement the algorithm differently according to the representation of the TSP problems. By analyzing the results, we will discuss the efficiency of the VTGA working with different implementations.

The data of the TSP problems used in the thesis is obtained from the TSPLIB, http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/. The information about the instances used in the thesis is presented in Table.

Table 1: Information about the TSPLIB instances

Instance Name	Best value
Bays29	2020
Swiss42	1273
Berlin52	7542
Eil76	538
Ch150	6528

2.9 FOREST PLANNING PROBLEM

The goal of solving a forest planning problem [19] is to find the best valid harvest schedule maximizing the even-flow of harvest volume. In a valid schedule, no adjacent forest stands are harvested in the same period. Attempting to generate a perfect even-flow schedule is easy within a linear programming environment, where decision variables are continuous. But when integer variables are used, there is no way we could find a valid equally good schedule. So we use that maximal even-flow as the target volume and try to find a valid schedule minimizing the sum of squared errors of the harvest totals from the target volume. This objective function f is defined as:

$$f = \sum_{k=1}^{z} (T - \sum_{n=1}^{d} a_n h_{n,k})^2$$

where z is the total number of harvest periods available to be assigned. T is the target harvest volume of each time period, a_n is the number of acres in stand n, $h_{n,k}$is the volume harvested per acre in stand n in the harvest period k, and d is the number of stands. A stand can be harvested only in one harvest period or not harvested at all, so $a_n h_{n,k}$ is either the volume of a stand n in period k or zero, if that stand is not scheduled for harvest in any period.

The 73-stand Daniel Pickett Forest and the 625-stand southern forest [19] are used in our study. The relevant input information could be found under "Western US example forest" and "Southern US example forest" at http://www.warnell.forestry.uga.edu/Warnell/Bettinger/planning/index.htm. For every stand, three time periods and a no-cut option are available. The target volume for the 73-stand harvest schedule is 34467 MBF (thousand board feet) and the target volume for the 625-stand forest is 2,972,462 tons.

CHAPTER 3

INTRODUCTION TO THE VIRUS TRANSMISSION GENETIC ALGORITHM

3.1 MOTIVATION AND PROJECT GOALS

With the traditional Genetic Algorithm, premature convergence, caused by loss of diversity, is always a problem preventing the GA from finding global optimal solutions. To help overcome this problem, by preserving diversity, the Island GA was proposed, which runs multiple populations with a communication structure. However, despite the special requirement for parallel machines, there is no evidence that shows the Island GA actually explores more regions than the traditional GA.

With progress of evolutionary biology, people realized it is insufficient to use only Darwinism to explain all phenomena in evolution. Therefore, new evolutionary theories have been proposed. Meanwhile, in computer science, new evolutionary algorithms are created based on those new concepts in biology. The Virus-evolutionary GA (VEGA) is such an algorithm focusing on the reverse transcription ability of a virus. But there are two problems with the VEGA. Firstly, the representation of a virus is a substring of a regular solution. But in some problems, good schemas are not kept within neighboring genes. And since a virus is not a complete solution, VEGA uses a relative evaluation function for virus individuals. To calculate the fitness of one virus, we need to evaluate every related host before and after infection. For problems with complex objective functions, that means expensive computational effort. In [9], the authors claim that with a certain number of evaluations VEGA performs better than a steady state genetic algorithm. However, they didn't tell whether that number includes both the

evaluations of hosts and the evaluations of viruses or just the evaluations of hosts. And it is very important to determine the performance of VEGA, because in VEGA one evaluation of a virus actually requires several evaluations of hosts.

In this thesis, we propose a new algorithm, called the Virus Transmission Genetic Algorithm (VTGA). Like we described in the introduction, the VTGA has three special features, the parasite-immediate selection, geographical isolation and the changing of virus statuses simulating the infection transmission model. There are three goals we want to achieve with the VTGA,

1. The algorithm is capable of finding the best solution.
2. The algorithm preserves diversity well.
3. The algorithm must be computationally affordable.

To conduct analysis on whether the algorithm satisfies our goals, we apply it to a 73-stand forest planning problem and demonstrate the effectiveness of this algorithm with several experiments.

3.2 VIRUS TRANSMISSION GENETIC ALGORITHM ARCHITECTURE

The VTGA is an algorithm that mimics the evolution of immune defenses against infectious diseases. To model the evolution, one host population and one virus population are defined. The outline of the algorithm is this: There are attacks from virus individuals to hosts. Weak hosts are killed by viruses and replaced by stronger hosts. As a result the overall fitness of the host population is gradually improved. To simulate this process, in the VTGA, we need to implement virus infections, virus spread and virus evolution inside a host (Figure 5). For infection, a virus selects a weak host and, by adaptation, the virus adopts information from its host. But a virus

won’t stay inside one host forever. In an appropriate situation, the virus escapes from its current host and selects another one. Virus spread simulates this process of escaping. Sometimes there could be many virus individuals infecting one host. That creates an environment for evolution among several viruses. With simple immune defense implemented in the VTGA, a weak virus, which has fitness worse than its host, won’t cause any change to its host. On the other hand, if a virus is good enough to defeat the defense of its host, then there is a chance that the host’s solution will be replaced by the solution of the virus, modeling the parasite-mediated selection. Eventually weak hosts in the population are replaced one by one so that the average fitness of the whole population is gradually improved. All these features make the VTGA a unique heuristic search method different from others.

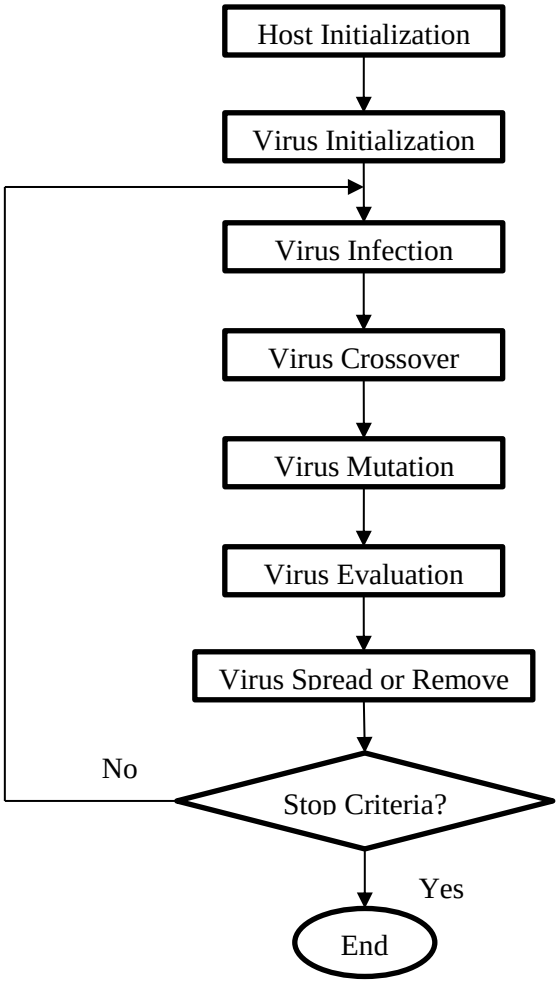


Figure 5: Flow Chart for a Virus Transmission Genetic Algorithm

3.3 HOST POPULATION AND VIRUS POPULATION

In biology, there are many several kinds of infectious pathogens, virus, and bacteria and so on. But to simplify our model, the VTGA uses virus to stand for all pathogens. Hence we model our algorithm with one host population and one virus population. Although named differently, host individuals and virus individuals have no difference in representation. Every individual, no matter a host or a virus, has a complete sequence representing a regular solution. At the beginning, the VTGA initializes hosts and viruses by assigning random solutions to them. Individuals of both populations have one fitness value generated by the same evaluation function.

That makes the VTGA different from a coevolutionary algorithm which uses different objective functions for different species. Being identical morphologically, virus individuals and host individuals differ from each other in behavior. The host population is mainly used for recording good solutions found during computation. No operation is defined for host population and there is no interaction among hosts. Virus individual, on the other hand, is designed to perform all the operations. In order to describe the algorithm better, from now on we use examples from the Forest Planning Problem to explain it. One host individual and one virus individual are listed below

Host:

121220223311231322022313223122222332111203311122233323331222211333323111 2

Fitness: 9960546

Virus:

121213223312332300302323223111212233212120331111221233233112221133332311 11

Fitness: 10253818

3.4 OPERATORS

There are two types of operators (Figure 6), intra-host operators and inter-host operators. Intra-host operators control evolution inside hosts, maintaining interactions among viruses infecting the same host. In our study, we implement crossover and mutation as two intra-host operators. The other kind of operators, inter-host operators, control interaction between a virus and a host, like infection and spread. Unlike the island model, distributed GA which exchanges information by migration, the VTGA exchanges information by letting viruses carry schemas of different hosts. For this purpose, we need infection and spread to exchange information between

virus and host. Meanwhile they are also responsible for moving a virus from one host to another. In the following sections, we are going to discuss details about operators.

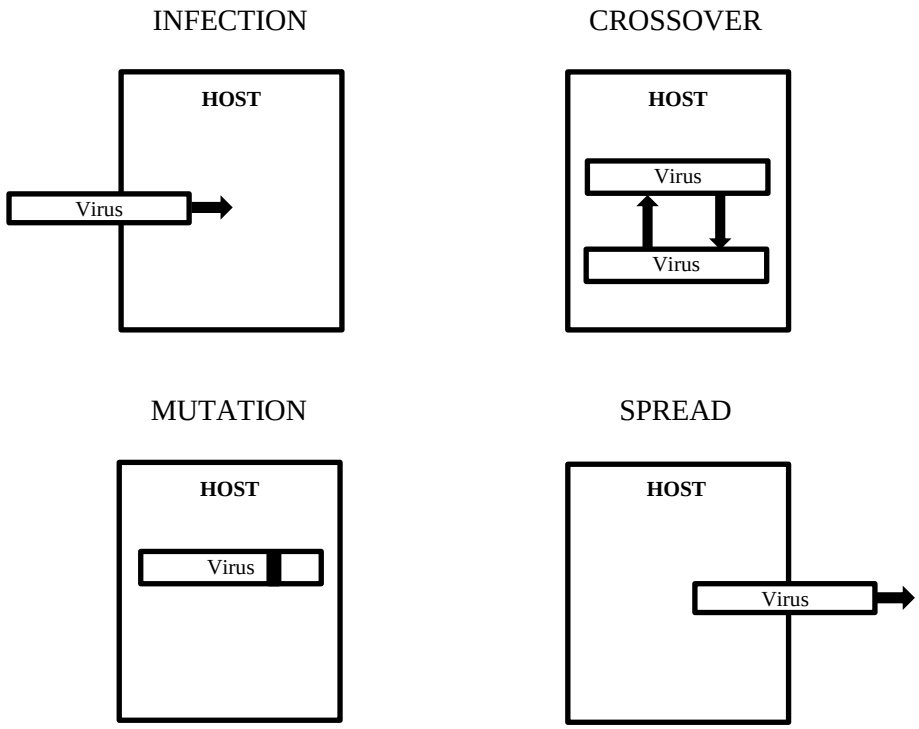


Figure 6: Outline for the VTGA

3.4.1 INFECTION

In reality, a virus can only replicate within living cells of an organism, in our case, a host. That means although a virus could survive a long time outside of a host environment waiting to infect its next host, viral evolution only happens inside a host. To simulate these features of a virus, in the VTGA, the virus can only have crossover and mutation when they are bonded to one host (Figure 7). So we need a process of infection to decide which viruses infect which hosts. This process includes two stages, host selection and sequence copying. To implement host selection, we use a tournament selection function to randomly pick up a few candidate hosts. The candidate

host with the worst fitness value will be selected as the target, following the fact that people with weak immune systems are likely to get sick.

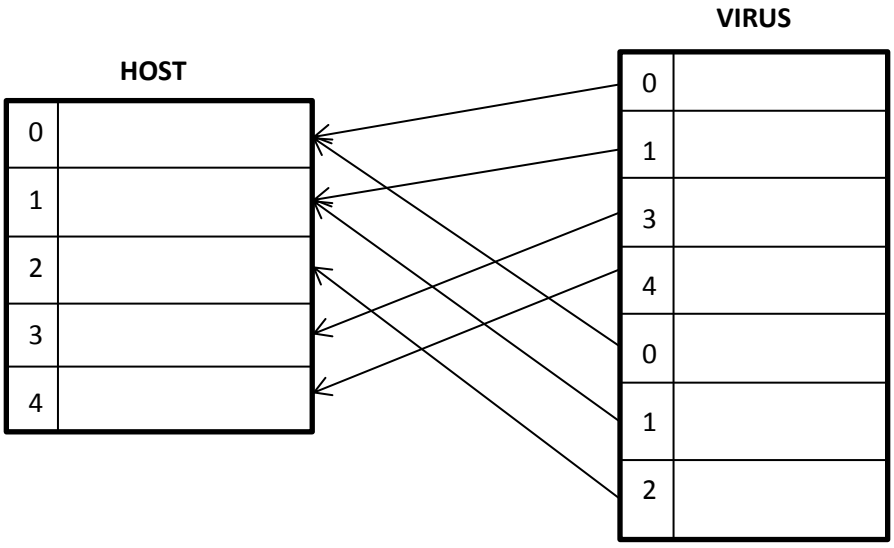


Figure 7: Graph for Virus Bonding Host

For example, in the FP problem, we have one virus V1, which tries to infect a host. The VTGA randomly selects three candidates, H1, H2 and H3 from the host pool.

V1: 12121322331233230030232322311212233212120331111221233233112221133333231111

Fitness: 10253818

H1: 12122022331123132202231322312222233211120331112223332333122221133332311112

Fitness: 9960546

H2: 12122322331121312202230322311323233111123301222221332312113123133332311111

Fitness: 10827408

H3: 32123222331123231000232323211212323212230231111123332333122221333333311112

Fitness: 10935573

In this case, H3 is the host to be infected by the virus V1.

Before we talk about sequence copying, let's take a look at what happens after viruses enter hosts' system in biology. At the early stage after invasion, the virus starts replicating and mutating to generate many strains. If there is one of them which beats the immune system, this invasion has succeeded. This virus is likely to cause sickness. Otherwise, the immune system eventually kills all strains and recovers from the infection. This is an adaptation process of the virus. But in this study, we don't want to generate hundreds or thousands of strains after every infection in our algorithm. So we simplify this process by copying information from hosts. Therefore, after selection, the VTGA copies several genes from the infected host and uses them to rewrite the sequence of the newly entered virus. In this case, a new sequence is formed by combining schemas from the current host and previous hosts, which are all good solutions found by the algorithm (Figure 8).

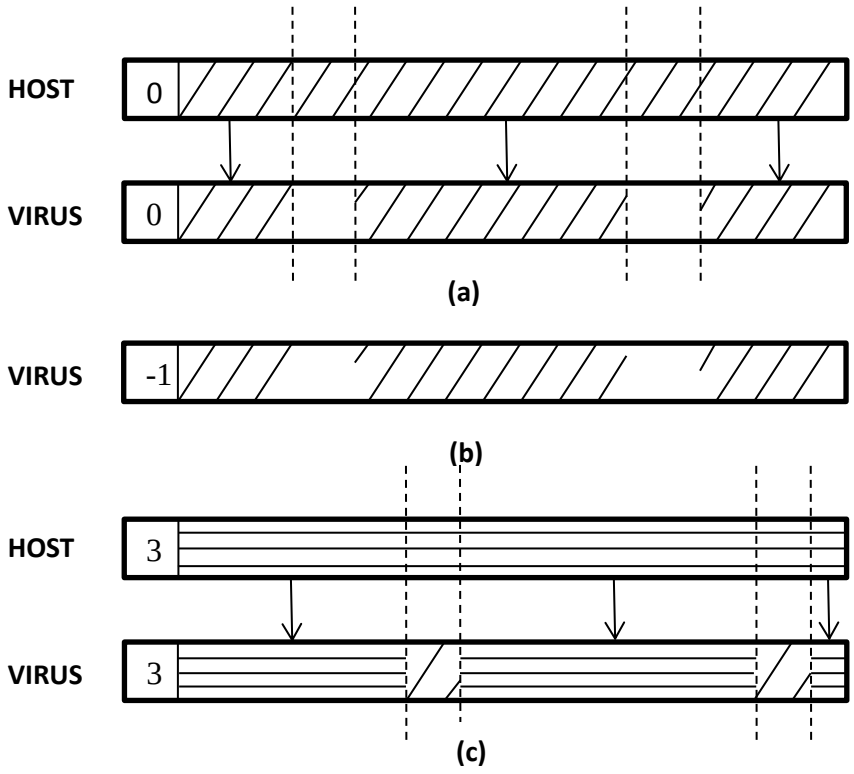


Figure 8: Diagram of Infection: (a) After infection, a virus copies several regions from its host. (b) At some point, a virus escapes from a host and wait to infect next host. -1 means the virus is free. (c) When a virus infects another host, it carries information from previous hosts.

Like operators of all heuristic search methods, information copying could be implemented in different manners. For the FP problem, we use a scheme similar to uniform crossover in the genetic algorithm. For every position of the host, the VTGA draws a random number between 0 and 1. If the number is smaller than a parameter called INFECT_RATE, we decide that position should be copied. We pick up several positions from the host in this way and the values of these positions are copied to the virus's corresponding positions. In our example, virus V1 is infecting host H3. Assume positions 1, 3, 4, 5, 8, 11, 15, 19, 22, 27, 33, 34, 40, 45, 48, 51, 54, 60, 65 and 70 are picked.

Before copying,

H3: 3212322233112323100023232321121232321223023111112333233312222133333331112

V1: 1212132233123323003023232231121223321212033111122123323311222113333231111

After copying,

H3: 3212322233112323100023232321121232321223023111112333233312222133333331112

V1: 3212332233123323000023232221121232321213033111121333333112221133332311111

After infection, a virus starts developing itself by recombination and mutation. After it "grows" inside its host for a while, it escapes and infects another host repeating the whole infection process.

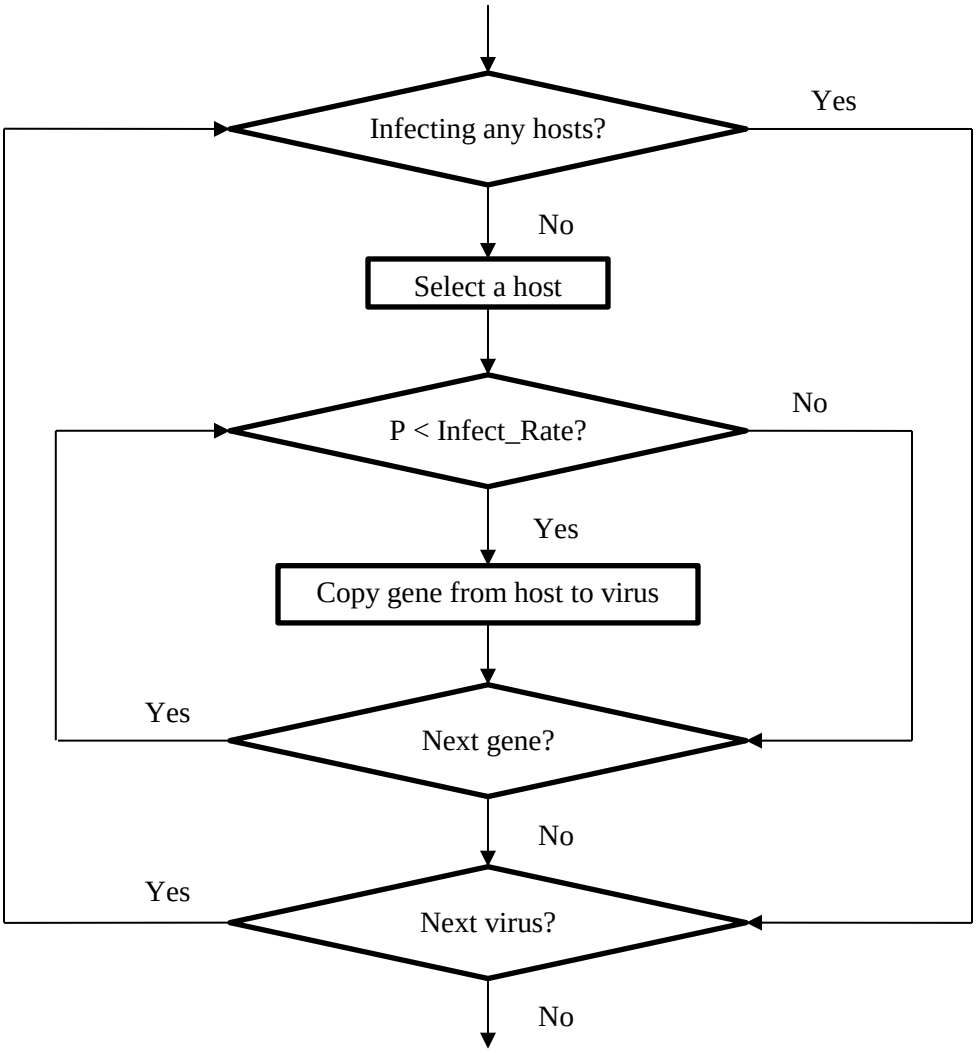


Figure 9: Flow Chart for Infection of the VTGA

3.4.2 CROSSOVER

In the Island GA, several genetic algorithms run simultaneously. Although information is exchanged with migration, the subpopulations of different GAs are actually independent. Individuals always belong to only one GA. Therefore, in the Island GA, selection and crossover are not much different from those of a simple GA. However, in the VTGA, viruses are moving from host to host. Although the total number of viruses remains the same through one

computation, the virus sizes inside hosts are changing all the time. Figure 10 shows what happens to the virus sizes inside different hosts. When we solve the Forest Planning Problem, we take a few snapshots of one computation of the VTGA at 2000, 5000, 8000 and 11000 generations, respectively. Every black bar in the graph represents the number of viruses inside one host, whose fitness value is indicated by a blue point. There are two things we can learn from Figure 10. Firstly, in one generation, the virus sizes are different in different hosts. Secondly, when a host has a bad fitness value (higher is worse in the case of the FP problem), there are more viruses inside the host generally, like we introduced in last the section about the infection operator.

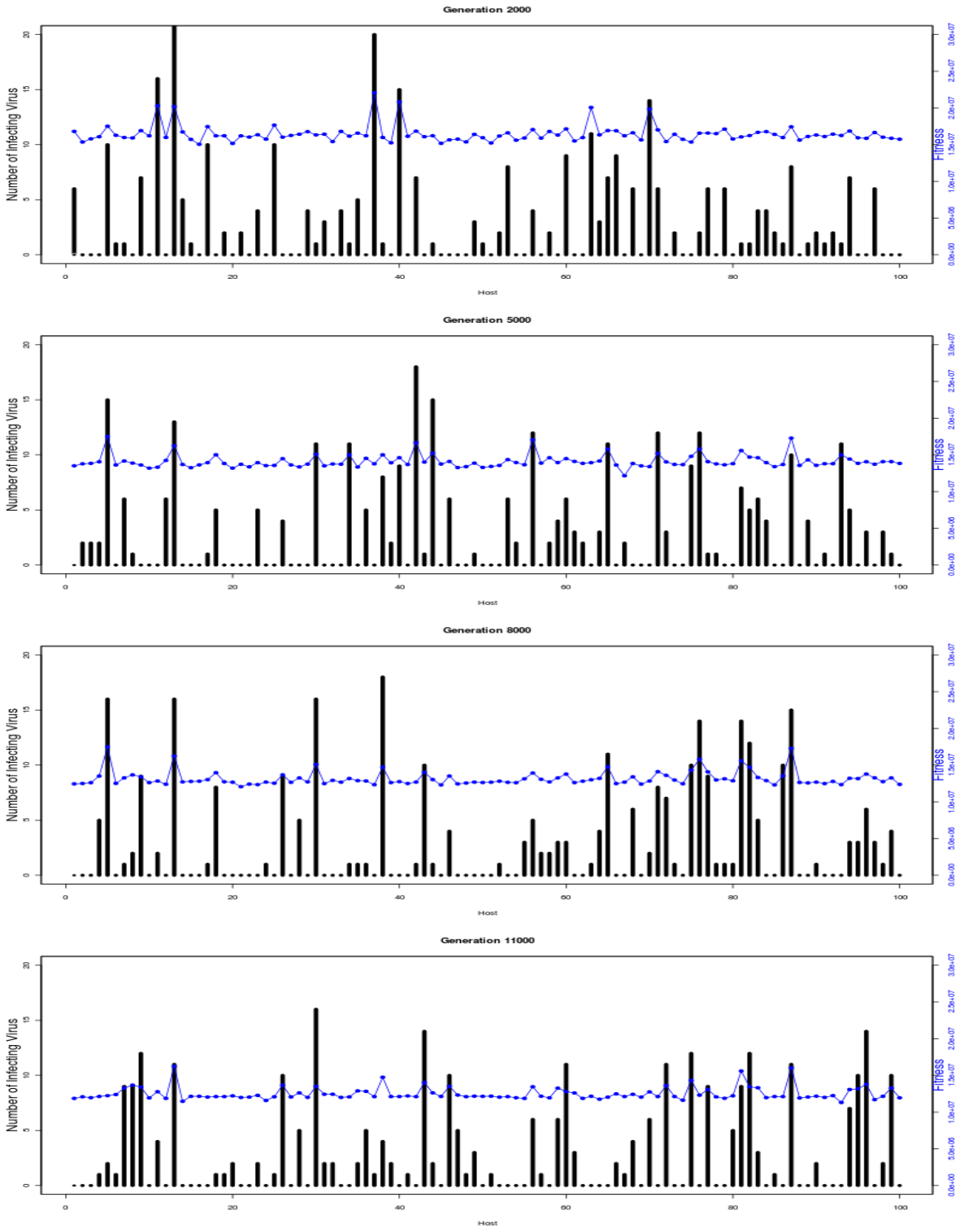


Figure 10: Diagram for numbers of Viruses infecting different Hosts.

Traditional crossover operators, especially those for the generational GA, usually require parent selection and replacement. However, it is too complicated to implement selection and replacement on dynamic populations in the VTGA. The reason is that with parent selection a list is required to put all parent individuals selected together. But in our algorithm the operator has to follow the concept that only viruses infecting the same host could be matched together for crossover. Therefore, before selection, we need to divide viruses into different groups based on their hosts. Then we need to do a parent selection for every one of these groups. And we need to keep track of these groups till the crossover ends. Finally the operator replaces the members of each group with new generated individuals. This whole process is too difficult for implementation, because in order to store those groups of viruses, we need to have dynamic arrays having different sizes in our computer program. That means we need to delete old arrays and create new arrays repeatedly. Besides, we need to do selection and replacement on these arrays and use the generated virus offspring from these arrays to replace the existing virus individuals in the original virus population. For these reasons, we want a different crossover method that doesn't have selection and replacement.

To do that, let's review what really happens with the selection and crossover operators of the GA. In the generational GA, we select good solutions and recombine them to create new individuals, forming a new generation. In the new generation, those individuals with good fitness are likely to be selected for recombination. On the other hand, those weak individuals with bad fitness will be eliminated by the selection process, because they have low probability to be picked by the algorithm. Judging from the consequence, information and patterns are mainly transferred from good individuals to bad individuals because good individuals stay in the population longer and have more opportunities to contribute schema to offspring than bad

individuals. In some occasions, it is possible that bad individuals have good patterns, but the chance is relatively small. When the population size is large like that of the traditional GA, it's nice to allow bad individuals to contribute patterns. But in the VTGA, the number of viruses inside a host is relatively much smaller, so we sacrifice the patterns of weak viruses by only allowing information to transfer from a good virus to a bad virus but not the other way around. Therefore, in the VTGA, we replace the traditional selection process with enumerating every pair of two virus individuals inside the same host. And we do the crossover based on the result of the comparison between the fitness values of the two viruses. The crossover process is like this: The better virus copies some genes of its sequence to replace the corresponding genes of the worse virus. How many positions are copied is controlled by a parameter called CROSSOVER_RATE. After the weaker virus is rewritten, the newly generated virus will not be evaluated and its fitness value remains the same. All individuals will be evaluated after crossover and mutation finish. We gain two advantages from this method. On one hand, inside the same host, the best virus has influence on every other virus. The second best virus has influence on every other virus except the best one and so on. On the other hand, the worst virus will be affected by all other viruses. And the second worst virus will be affected by all other viruses except the worst. That means in the VTGA crossover, only good solutions could have an influence on bad solutions. However, bad solutions won't cause any impact on good solutions. With the common GA, this is considered as greedy and likely to cause dominance of several outstanding individuals. But with the VTGA, we have a host population to keep the solutions diversified, so we reduce the risk of being trapped in local optima. Let's discuss a little further with an example. In one host, suppose we have two virus sequences, S1 and S2 as follows,

S1: 113203203211221203202313222113332323111203211122323323123221333333322 12

Fitness: 8948552.432628956

S2: 221203223211221203202313222113332332111203211121323323232132213333323311 1

Fitness: 8535594.490528988

We learn that S1 is worse than S2. So we randomly select some positions from S2. If those positions on S1 have different values, rewrite them with the values of S2.

S1: 123203203211221203202313222113332323111203211121323323232232213333323211 2

S2: 221203223211221203202313222113332332111203211121323323232132213333323311 1

After the change of information, the fitness values remain. So the fitness value of S1 is still 8948552.432628956 and the fitness value of S2 is still 8535594.490528988. They won't be changed until we finish the crossover process and the mutation process.

3.4.3 MUTATION

To implement a mutation operator, the VTGA follows the manner of the GA. In our study for the forest planning problem, the VTGA reassigns random values to genes with a low probability and this process is controlled by a parameter named MUTATION_RATE.

3.4.4 SPREAD AND RECOVERY

In the VTGA, a virus doesn't infect the same host through the whole evolution. After a few generations, the virus escapes from its current host and infects another one. The spread operator simulates the process of escaping. According to the infection transmission model discussed in the previous chapter, after infection, the host either recovers or dies from the infection. So with the spread operator of the VTGA, we consider two conditions. If the fitness value of a host is worse than that of a virus infecting it, the host sequence is probably, but not necessarily (controlled by a

probability), replaced by the virus sequence. Let us explain this process with more details. If a host's immune system is weaker than the virus, then the virus is probably going to kill the host. In that case, the host will be replaced by a stronger host which is able to survive the virus infection. Therefore, we copy the chromosome of the virus and use it to replace the host's chromosome. For example, a host sequence and one of the viruses infecting it are listed below,

Host:

12122022331123132202231322312222233211120331112223332333122221133332311 12

Fitness: 9960546

Virus:

22120322321122120320231322211333233211120321112132332323213221333332331 11

Fitness: 8535594.490528988

Since the virus beats the host, therefore, the VTGA replaces the host with the virus so that the host individual becomes stronger.

Host:

22120322321122120320231322211333233211120321112132332323213221333332331 11

Fitness: 8535594.490528988

Virus:

22120322321122120320231322211333233211120321112132332323213221333332331 11

Fitness: 8535594.490528988

However, we need to highlight that this replacement doesn't happen all the time. In biology, sometimes extremely virulent viruses occur inside hosts. These viruses are so strong that they kill their hosts quickly before they have a chance to spread themselves. In another words, we moderate the algorithm so that it becomes less greedy by not recording all the good solutions it

generates in the host population. To control the process, a parameter, the SPREAD_RATE, is used.

Another condition is when a virus is not better than its host. For a virus weaker than its host, the VTGA allows it to escape. The probability is controlled by a parameter called the RECOVERY_RATE. Once a virus escapes from its host, we would consider the host's immune system to have recovered from the infection of this virus so that the virus won't perform crossover or mutation until it infects next host.

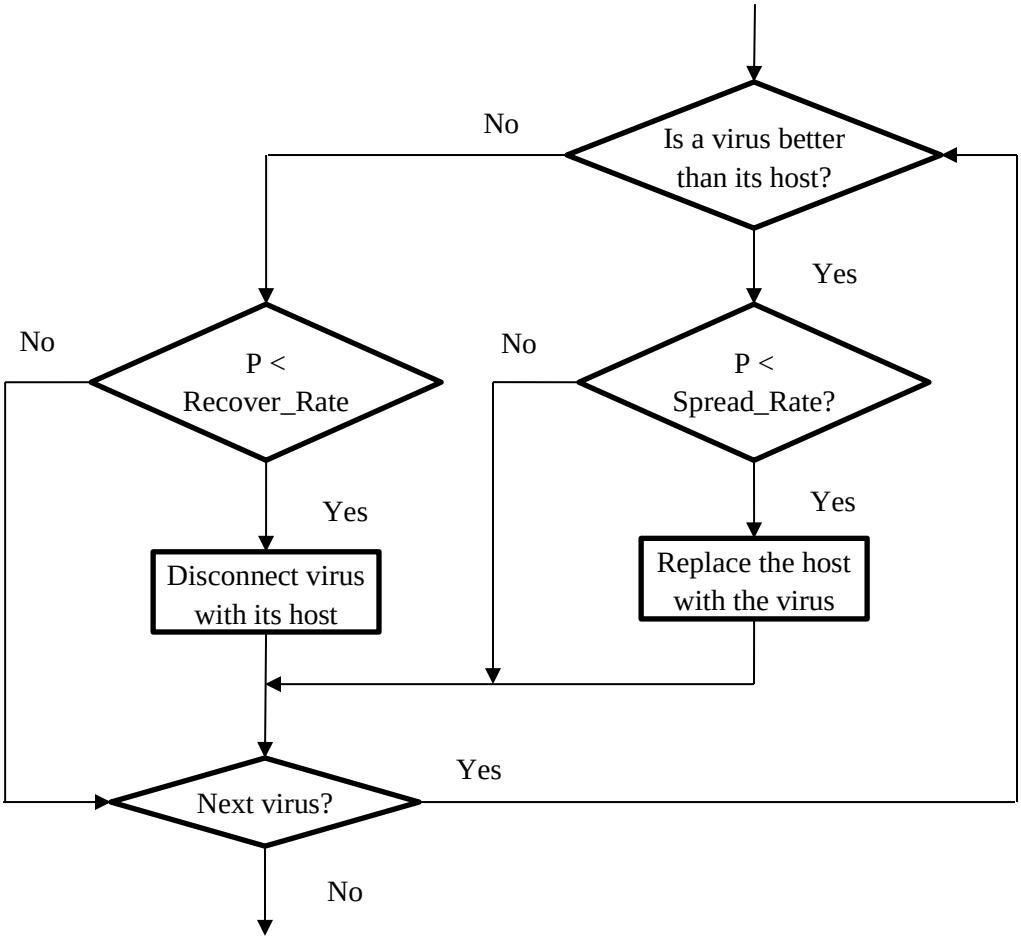


Figure 11: Flow Chart for Spread and Recovery

3.4.5 LOCAL BEST FITNESS MASKING

In the VTGA, the infection operator randomly selects a few candidates from the host population, and a virus will infect the weakest one of the candidates. This process simulates the fact that people with weak immune systems are likely to get sick. So in the host population, generally speaking, the worse a host is the more opportunities it has to get infected. But there is one problem with this scheme. Suppose VTGA selects N candidates from the host population in every infection. So the best N-1 host individuals will never be selected. In this case, VTGA spends all its computational effort on improving relatively bad solutions. This mechanism gradually raises the lower bound of all host fitness in the whole population. However, in optimization problems, this is not efficient because the goal is to find the best solution. So we use a method called fitness masking on the best host. At the end of every generation, the algorithm goes through all host individuals and finds the best individual in the host population. After the VTGA records the best individual, it randomly selects another individual from the host population and uses the fitness value of the newly selected host to mask the fitness value of the best individual in the population.

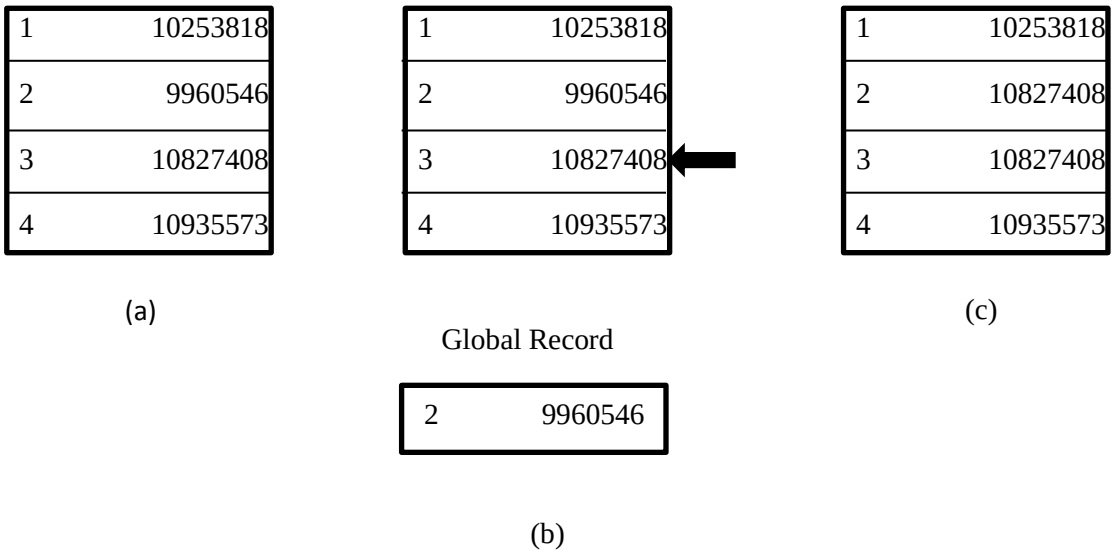


Figure 12: Graph for Fitness Masking: (a) In one simple example of four host individuals of the 73-stand FP problem, suppose the VTGA selects three candidates for infection. Then individual 2 will never be selected. (b) With fitness masking, at the end of a generation, the best individual is saved in a global record. Then the VTGA randomly selects another host (indicated by the arrow). (c) Finally the algorithm assigns the fitness value of host 3 to cover the fitness value of host 2.

CHAPTER 4

EXPERIMENTS AND RESULTS

4.1 EXPERIMENTS

Like the traditional GA, parameters control the balance between exploration and exploitation. The VTGA has seven parameters, Host Population Size, Virus Population Size, Crossover Rate, Mutation Rate, Infection Rate, Spread Rate and Recovery Rate. Since the VTGA is a new algorithm, the effect of these parameters is unknown. So we make it our first mission to figure out the best parameter setting through experiments. Because Kubota et al used the generalized Rastrigin function to study the VEGA [9], for the purpose of comparing two algorithms, we use the same problem to study the configuration of the VTGA. The generalized Rastrigin's function is as follows [18],

$$f(x) = nA + \sum_{i=1}^{n} x_i^2 - Acos(2\pi x_i)$$

where A = 10; n controls the number of dimensions; $-5.12 \leq x_i \leq 5.12$. The global minimum of this function is 0. In [9], the VEGA was tested with this function where n = 5. To compare the result, we follow this setting of the function in our experiments.

After studying the parameters, we applied the VTGA to the 73-stand forest planning problem to further examine the effectiveness of the VTGA.

4.2 PARAMETERS

The VTGA has seven parameters. For every problem we solve with the VTGA, the first thing we need to do is to figure out the best configuration of parameters. In this section, we want to show

how we choose values for the parameters when we search for the optimal solution of the Rastrigin's function. We present here the results of the experiments we do for finding the best configuration and we discuss the meaning of increasing and decreasing the value of every parameter. After we find the best configuration, we compare the performances of the VEGA and the VTGA with the configuration.

Because the number of all possible parameter configurations is huge, we can't go through all of them to find the best. Therefore, we examine only one parameter at a time while other parameters are fixed with the values in the default configuration (Table 1). The default values of parameters are basically chosen at random except for the host size and the virus size. In the VTGA, the host population could be seen as a buffer storing good solutions found during a searching process. Therefore the host size depends on how many patterns a problem has. When the representation for a problem to be solved has more genes or has more possible values of every gene there are more possible patterns. If the host size is too small, the VTGA only remembers a few good solutions. In that case, not much diversity is preserved. On the other hand, if the host size is too big, many solutions will be saved even if some of them are not good enough. Therefore, the VTGA wastes much time on improving bad solutions. Compared to the FP problem, the generalized Rastrigin's function is relatively less complex, so we set a small host population size in the default configuration. Since evolution depends on the virus population in the VTGA, the virus population size is important to the effectiveness of the VTGA. By infection, viruses are tied to different hosts. Although how many viruses there are in every host is a different number, generally speaking, the bigger the virus population size is, the more viruses are in one host on the average. Since we need to compare every pair of viruses in the same host, if there are n viruses in one host, the VTGA needs to do $n*(n-1)$ comparisons. So we must be

careful with the virus population size to keep the VTGA running efficiently. In the default configuration, we use 300 as the virus population size, giving about 12 viruses to every host on the average.

For every parameter combination we are going to test, we run 10 trials of the VTGA. A trial stops when there is no improvement on the best solution in 100 generations. The generalized Rastrigin's function is a minimization problem, the global minimal is 0. If the fitness of a solution is closer to 0, the solution is better. After one trial, we record the fitness of the best solution and how many evaluations are required to find it. By comparing all these results, we want to find the best configuration of the algorithm and to investigate how performance reacts to the increase or decrease of different parameters.

Table 2: Default Configuration for the VTGA Experiment

Host Size	25
Virus Size	300
Crossover Rate	0.3
Mutation Rate	0.01
Infection Rate	0.9
Spread Rate	0.8
Recovery Rate	0.8
Stop Criteria	No Improvement in 100 Generations

3.2.1 CROSSOVER RATE

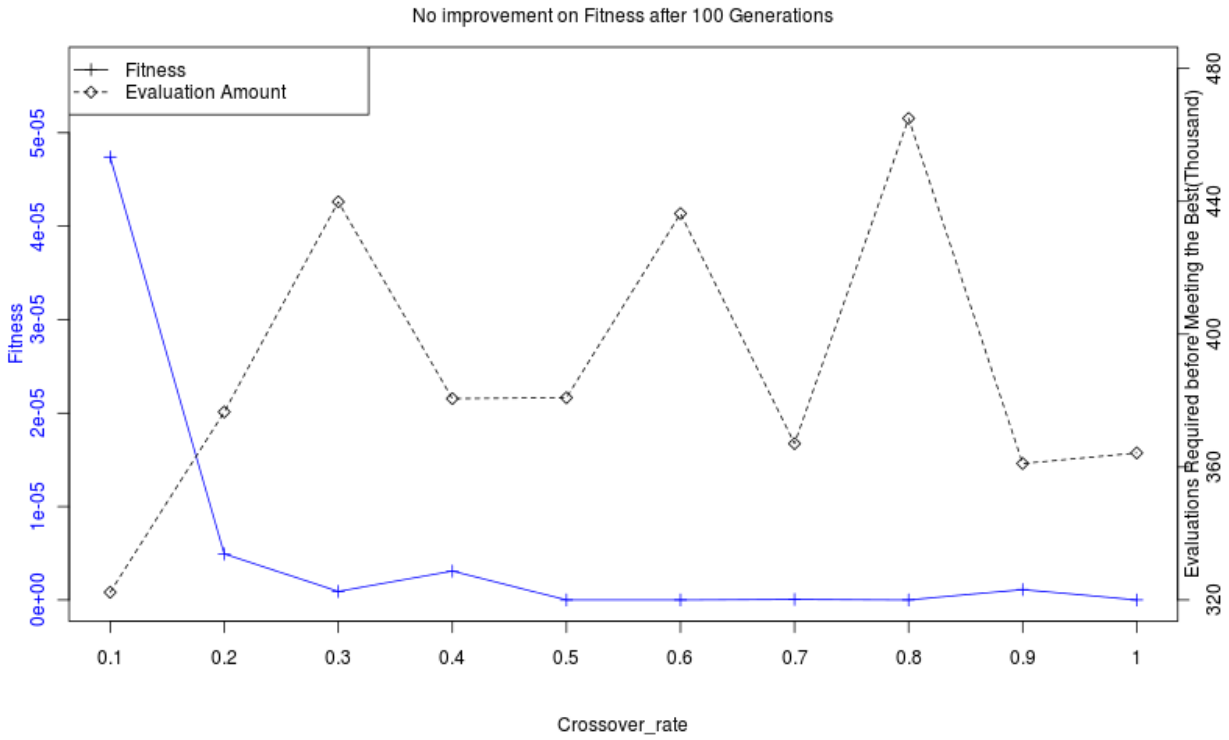


Figure 13: Result for Crossover Rate

Table 3: Result for Crossover Rate

Crossover Rate	The Fitness of the Best Found Solution	The Amount of Evaluations Required before Finding the Best
0.1	4.74042E-05	**322320**
0.2	4.92952E-06	376590
0.3	9.16891E-07	439860
0.4	3.09027E-06	380610
0.5	4.30825E-09	380880
0.6	5.65947E-09	436320
0.7	6.45951E-08	367080
0.8	2.77620E-09	464970
0.9	1.11332E-06	361080
1	**8.76606E-10**	364260

From the result of examinations on different crossover rates [Figure 13 and Table 2], Generally speaking, the VTGA performs well with a crossover rate ranging between 0.5 and 1 except for

0.9, if we consider only the fitness of the best found solution. But if we look at the numbers of evaluations required for finding the best solution, we learn that some of the crossover rates require more evaluations. In general 0.7 and 1 are two good configurations. They lead the VTGA to find good solutions and don't require too many evaluations. Although setting the crossover rate to 1 seems to be a better choice than setting that to 0.7, doing that means in the crossover process, the better virus will copy its whole genome to the weaker virus. In another words, after performing the crossover operator, all viruses in the same host become identical because they are all copies of the best virus inside the host. To encourage information exchanging, we choose the second best 0.7 as the best configuration for the crossover rate. Therefore, in the crossover process, the weak virus is able to preserve 30% of its own information instead of none.

4.2.2 MUTATION RATE

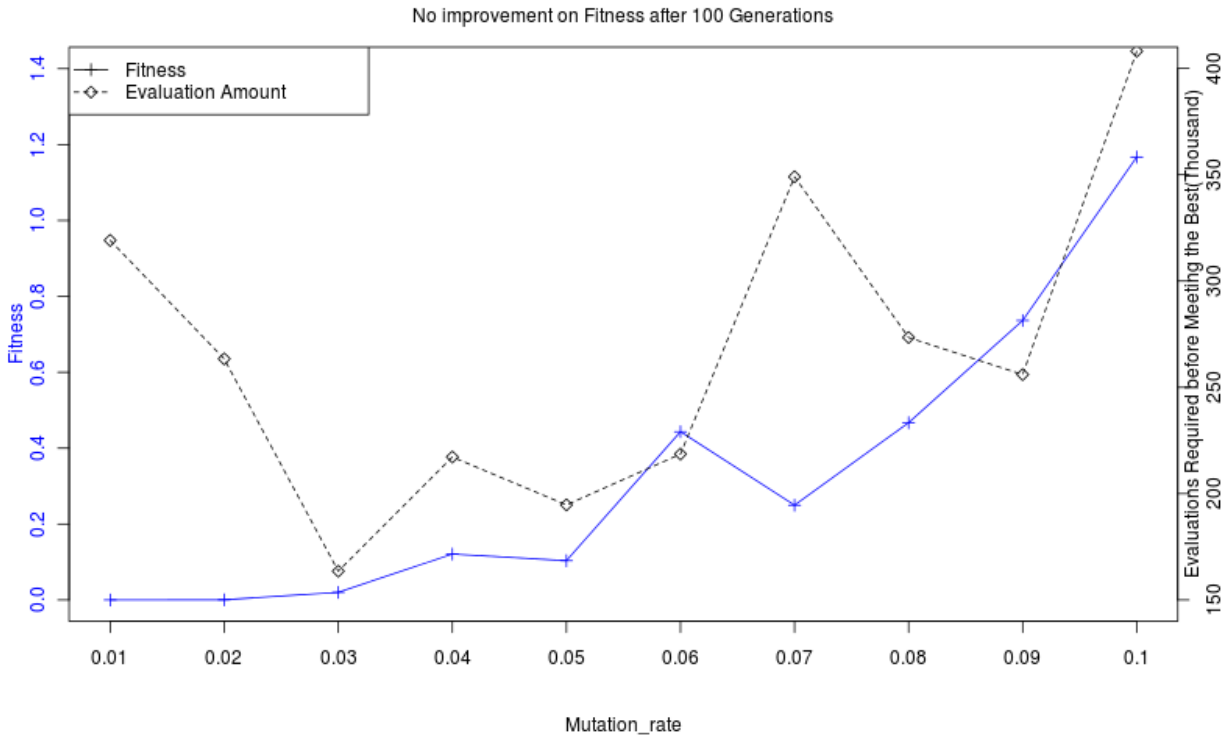


Figure 14: Result for Mutation Rate

Table 4: Result for Mutation Rate

Mutation Rate	The Fitness of the Best Found Solution	The Amount of Evaluations Required before Finding the Best
0.01	**2.72085E-07**	319140
0.02	4.91156E-04	263400
0.03	2.00210E-02	**163560**
0.04	1.20905E-01	217260
0.05	1.03526E-01	194730
0.06	4.43296E-01	218640
0.07	2.49691E-01	349050
0.08	4.67034E-01	273450
0.09	7.36423E-01	256020
0.1	1.16718	408090

In the GA, it is good to maintain a low frequency of mutation. Figure 14 and Table 3 shows that this guideline is also true in the VTGA. A high mutation rate means less exploration power and

more evaluations before finding the best solution. When we design the VTGA, our major goal is to find better solutions by using this algorithm. So a better fitness of the best found solution weights more than a lower amount of evaluations in our study. Therefore, we choose 0.01 as the best value of the mutation rate in the VTGA for solving the Rastrigin's function.

4.2.3 INFECTION RATE

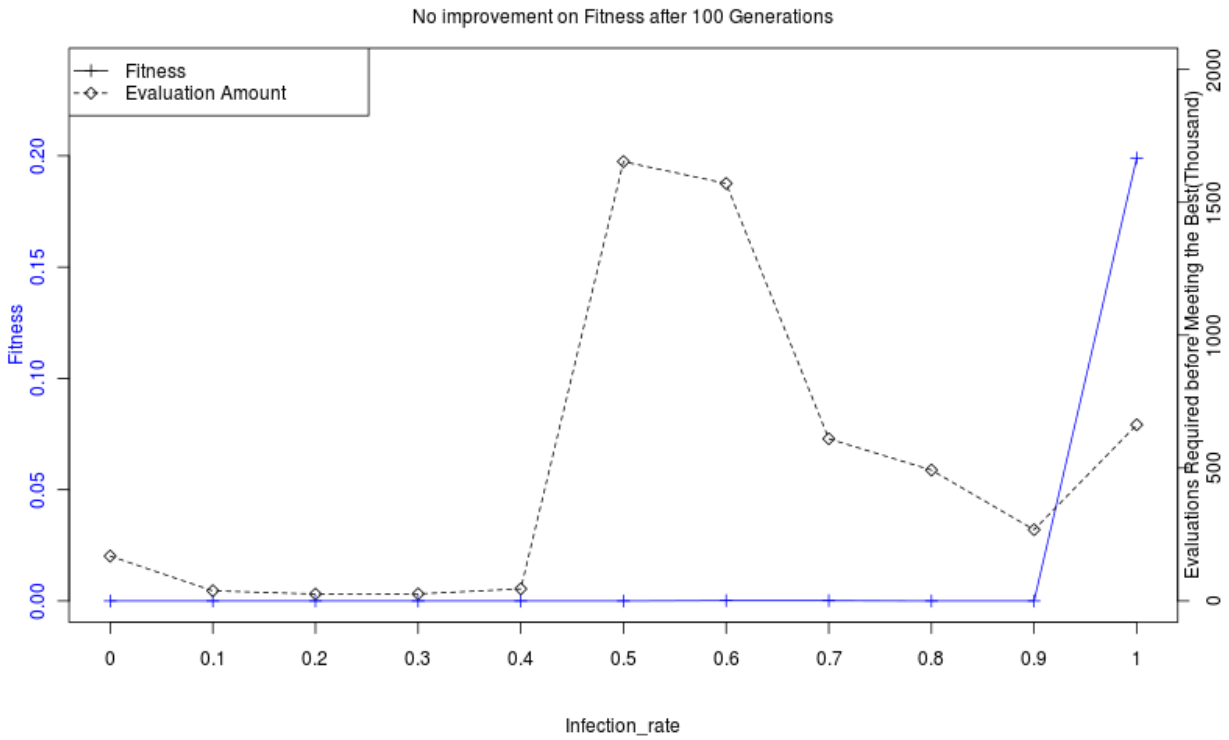


Figure 15: Result for Infection_Rate

Table 5: Result for Infection Rate

Infection Rate	The Fitness of the Best Found Solution	The Amount of Evaluations Required before Finding the Best
0	3.37508E-15	169020
0.1	**0**	38520
0.2	**0**	**26130**
0.3	**0**	26700
0.4	**0**	45720
0.5	7.15517E-13	1653180

0.6	1.84068E-04	1571070
0.7	1.70627E-04	610440
0.8	1.23027E-05	492690
0.9	4.75374E-07	268200
1	1.98993E-01	663090

The infection rate is a parameter that controls how much adaptation a virus needs after it infects a host. A high infection rate means that a virus copies a lot of information from its current host. So it becomes similar to the host after infection. On the other hand, a low infection rate means a virus is able to keep most of its own information. If we say that a virus does a local search around the solution of its host, then the infection rate, or the adaptation rate, actually controls the range of the neighborhood around the host where the virus is going to search. For example, suppose we have a host and a virus. Before the infection, let's assume every gene of the host has a different value from the same gene of the virus. So by the Hamming distance, we know that the distance between the host and the virus equals the lengths of their sequences, which is denoted by N. And let us assume the infection rate is α. So after the infection, the Hamming distance between the host and the virus would be approximately αN. In real searches, a host and a virus probably would have some genes with same values before adaptation. In this case, after the infection, the Hamming distance between the host and the virus would be smaller than αN. Overall, after the infection, the virus does local searches in the neighborhood within a range of αN from its host.

From Figure 15 and Table 4, we see that the VTGA prefers a low infection rate rather than a high one. With a low infection rate, the VTGA finds the global optimum of the Rastrigin's function quickly. When configured to between 0.1 and 0.3, the VTGA finds the optimal solution within much fewer evaluations than when it's configured to others. Based on this result, a question is proposed as to whether the VTGA performs even better without any adaptation. In

that case, after a virus infects a host, the virus doesn't obtain any information from the host. From Figure 15 and Table 4, we could see that if we set the infection rate to 0, which means no adaptation at all in the infection process, the VTGA performs poorly. It requires more evaluations before it has any improvement on solutions and the best solution it finds is not as good as that found by the VTGA with adaptation. Therefore, we could draw a conclusion that it is important to allow viruses to adapt to their hosts. But the adaptation should be limited to a low level. Since the VTGA is able to find the global optimum with the infection rate ranging from 0.1 to 0.3, we choose the value requiring the least evaluations, which is 0.2, as our best infection rate value.

4.2.4 SPREAD RATE

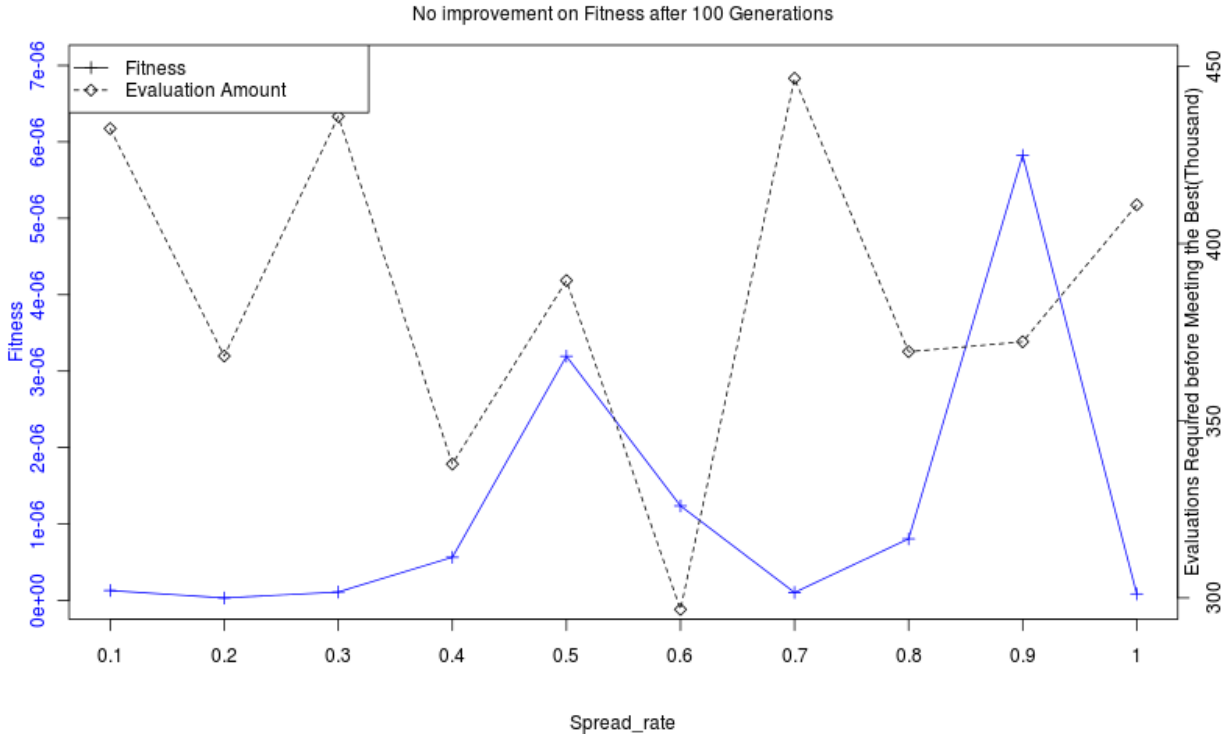


Figure 16: Result for Spread Rate

Table 6: Result for Spread Rate

Spread Rate	The Fitness of the Best Found Solution	The Amount of Evaluations Required before Finding the Best
0.1	1.28835E-07	432450
0.2	**3.10000E-08**	368280
0.3	1.09154E-07	435870
0.4	5.63466E-07	337800
0.5	3.19474E-06	389580
0.6	1.23655E-06	**296760**
0.7	1.02764E-07	446640
0.8	8.04891E-07	369510
0.9	5.82291E-06	372300
1	8.40811E-08	410940

Like we described in the previous chapter, the reason to introduce the Spread Rate in the VTGA is because in biology an extremely virulent strain of virus is likely to kill its host before the host has a chance to spread the strain out. With the death of the host, the strain doesn't have any contribution to the evolution of its virus species. In the VTGA, the Spread Rate controls whether a virus replaces its host or not when it defeats the host. A high value of the Spread Rate means the VTGA is likely to remember a virus when it is good enough to defeat its host. Generally speaking, this helps the VTGA converge to good solutions. But on the other hand, like the problem of the GA, premature convergence prevents the VTGA from finding the global optimum.

From Figure 16 and Table 5, the VTGA shows the best performance with 0.2 as the Spread Rate, that is why we make it our choice for the parameter configuration.

4.2.5 RECOVERY RATE

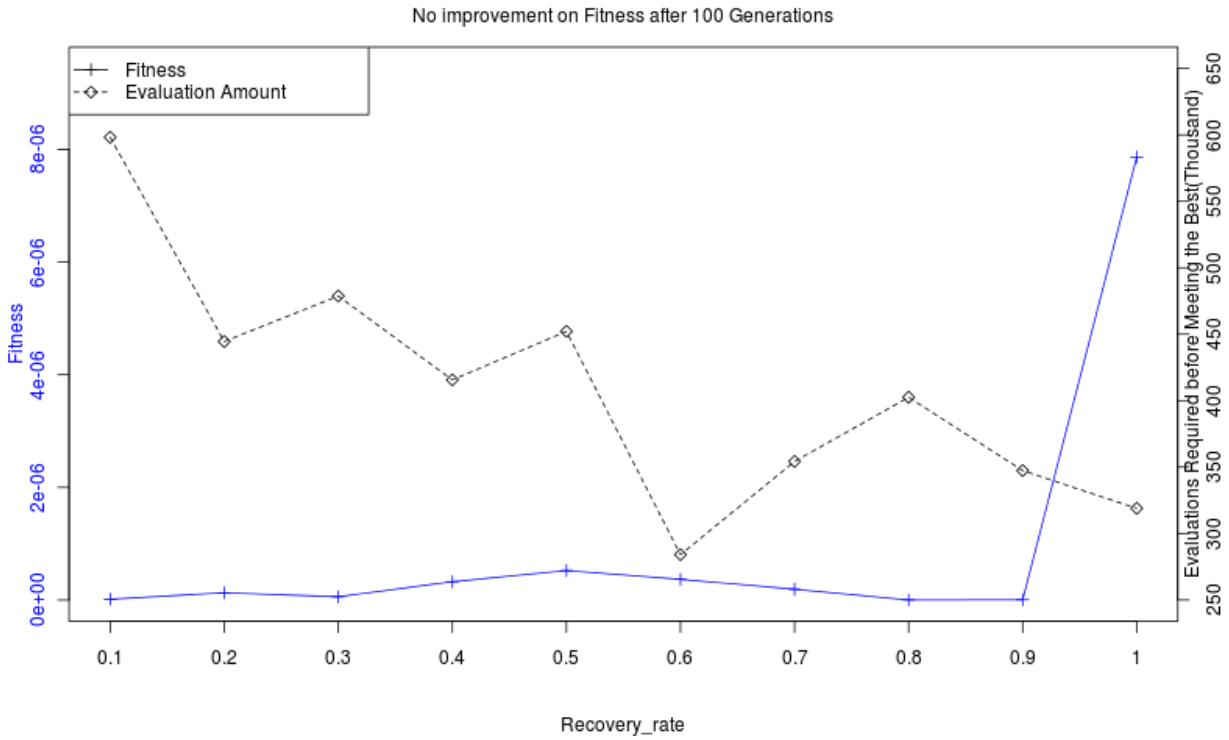


Figure 17: Result for Recovery Rate

Table 7: Result for Recovery Rate

Recovery Rate	The Fitness of the Best Found Solution	The Amount of Evaluations Required before Finding the Best
0.1	1.53804E-08	598200
0.2	1.28807E-07	444510
0.3	5.78854E-08	478740
0.4	3.25419E-07	415620
0.5	5.22849E-07	452160
0.6	3.67616E-07	**284130**
0.7	1.91739E-07	354240
0.8	**9.41668E-10**	402570
0.9	4.46030E-09	347400
1	7.86477E-06	318960

At the end of every generation, the algorithm has a comparison between the hosts and the viruses. If a virus doesn't have a better fitness than that of its host, there is a probability the virus will

escape from its current host. The recovery rate controls the probability. If the recovery rate is high, it means a virus is likely to escape from its host when the virus can't defeat the host. And that affects the amount of infection also, because the more often viruses escape from hosts, the more infections the VTGA has. Since the host population stores good solutions ever found by the VTGA, more infections mean more mixtures of found good solutions. On the other hand, if the recovery rate is lower, viruses stay inside their hosts longer, meaning more searches around the hosts. Therefore, the recovery rate actually controls a tradeoff between exploration and exploitation. From Figure 17 and Table 6, we find that the VTGA performs the best with 0.8 as the recovery rate. So we keep this setting in our best configuration.

4.2.6 HOST POPULATION SIZE

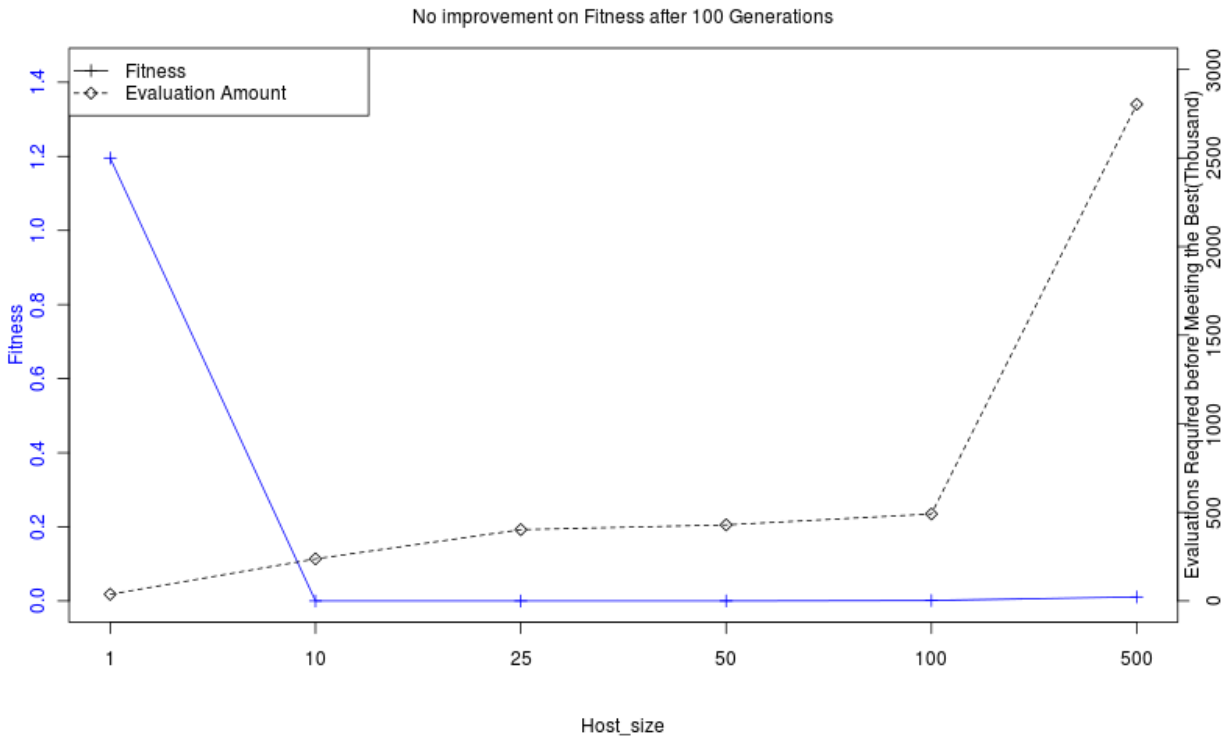


Figure 18: Result for Host Population Size

Table 8: Result for Host Population Size

Host Population Size	The Fitness of the Best Found Solution	The Amount of Evaluations Required before Finding the Best
1	1.19554	**37800**
10	**1.45906E-11**	237720
25	9.41668E-10	402570
50	2.21437E-04	429630
100	1.31831E-03	491130
500	1.09171E-02	2803200

In the VTGA, the host population is used for recording the best solutions ever found by the algorithm. When we use only 1 host in the population, the VTGA functions similarly as the GA where all individuals are placed together in the same pool. Judging from the result, the algorithm finishes quickly without finding good solutions. This evidence supports our idea about using a host population to record effective patterns of solutions and to maintain diversity. However, too many hosts are not really a good thing for the VTGA. Since we use the improvement on the average fitness of all host individuals as the stopping criteria, if we have more host individuals than we need, the algorithm will spend much effort on improving bad solutions. In Figure 18, we see that when the host population size increases, the number of evaluations increases as well. But the fitness of the best found solution gets worse. That means if the host size is too big, the VTGA remembers too many solutions where even some of them are not good enough and useless for finding the global optimum. In that case, the VTGA does a lot of meaningless work to improve bad solutions. Therefore, when we choose the host population size for the VTGA, we need to make it appropriate for the problem we want to solve. For the 5-dimension Rastrigin's function we need 10 hosts according to Figure 18 and Table 7.

4.2.7 VIRUS POPULATION SIZE

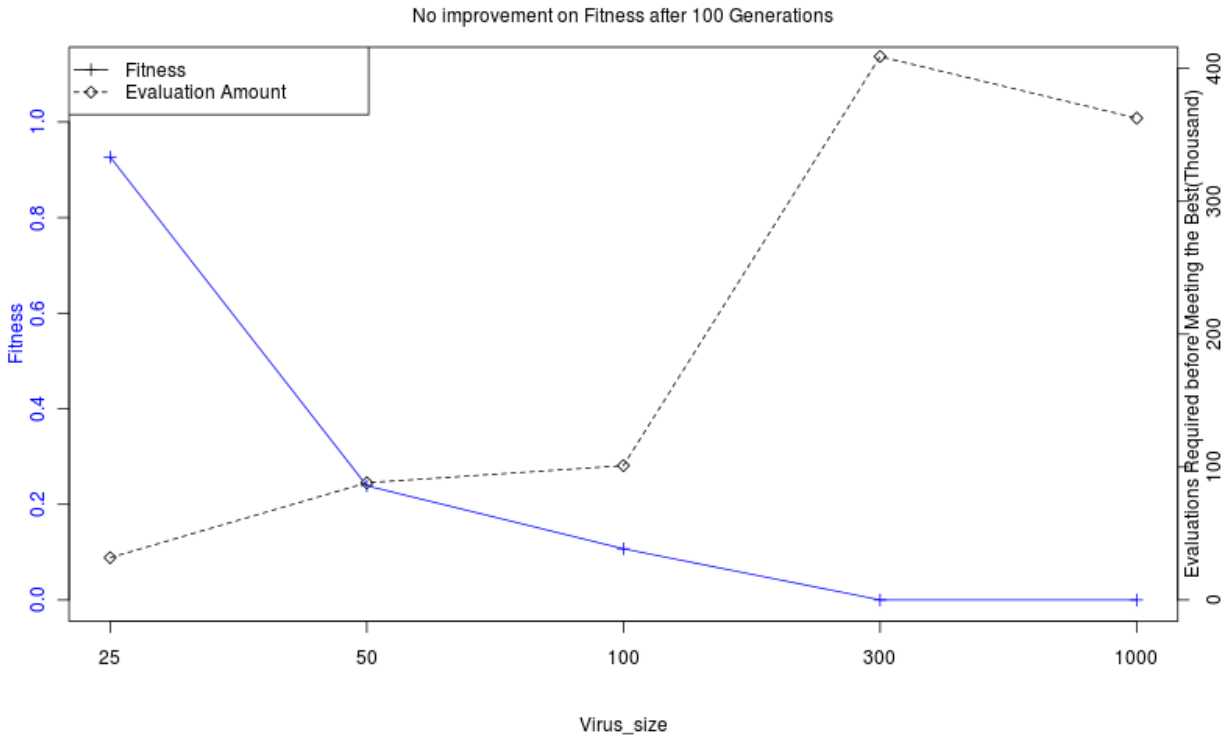


Figure 19: Result for Virus Population Size

Table 9: Result for Virus Population Size

Virus Population Size	The Fitness of the Best Found Solution	The Amount of Evaluations Required before Finding the Best
25	9.26668E-01	**31852.5**
50	2.38877E-01	88240
100	1.07015E-01	101120
300	1.17900E-06	408990
1000	**0**	362600

Figure 19 and Table 8 shows the result of the experiment testing different virus population sizes. The virus population is the driving force of the improvement in the VTGA. Instinctively speaking, more viruses anticipating crossover and mutation are beneficial to the VTGA's performance because more patterns are contained in the population. From the result, 1000 is the

best setting for virus population size, leading the VTGA to find the global optimum of the 5-dimension Rastrigin's function.

4.2.8 RATIO OF HOST POPULATION SIZE TO VIRUS POPULATION SIZE

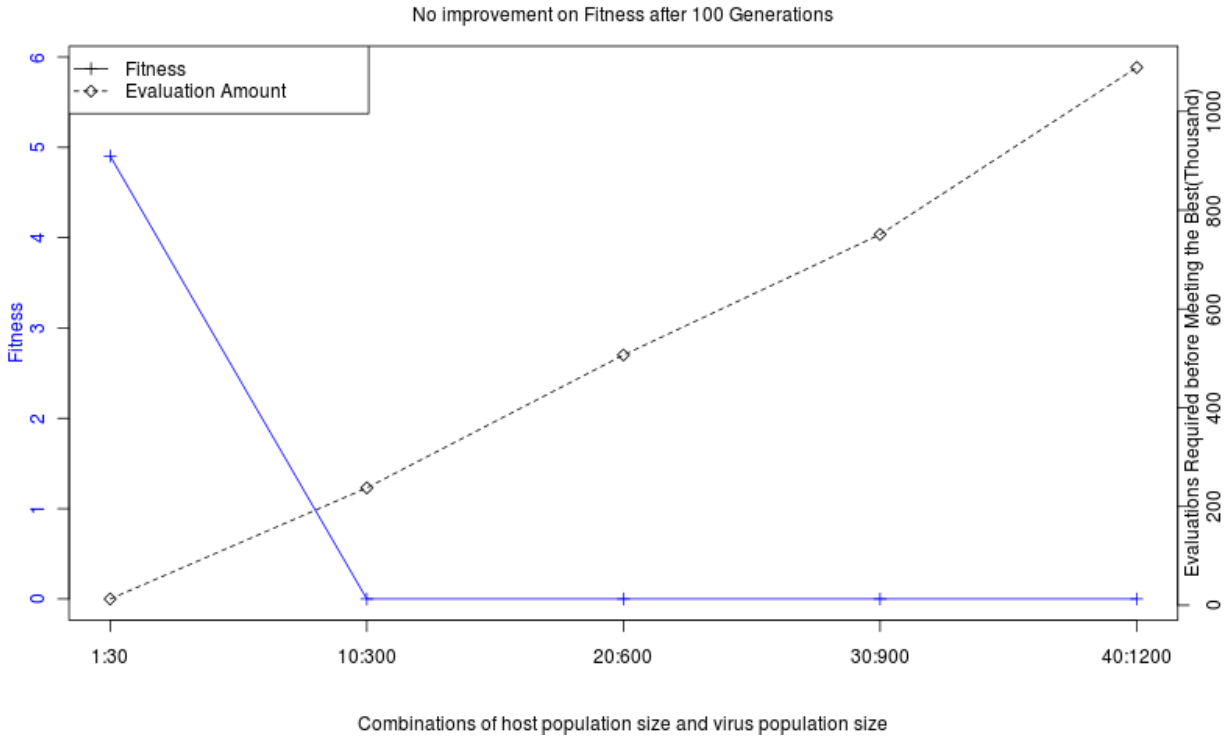


Figure 20: Result for the Ratio of Host Population Size to Virus Population Size

Table 10: Result for the Ratio of Host Population Size to Virus Population Size

Host Population Size	Virus Population Size	The Fitness of the Best Found Solution	The Amount of Evaluations Required before Finding the Best
1	30	4.90422	**12699**
10	300	1.45906E-11	237720
20	600	6.34159E-14	506700
30	900	**0**	750870
40	1200	1.59872E-15	1089000

After we study the best configuration of the host population size and the virus population size, we have another concern about whether the performance of the VTGA is related to the ratio of the host population size to the virus population size. In this experiment studying the best host population size, we decide that the best setting is 10 hosts for the Rastrigin's function while the virus population size is 300, which means there are 30 viruses improving every host on the average. Therefore, we want to test whether the ratio of 1 to 30 has any impact on the VTGA. So we create another experiment testing different combinations of the host population size and the virus population size (1:30, 10:300, 20:600, 30:900, and 40:1200). The ratios of all combinations are all equal to 1/30. Figure 20 and Table 9 show that the VTGA finds really good solutions with all of these combinations except the first one. The reason for that is probably the same as what we discussed in the study of the host population size. Using only 1 host doesn't keep diversity in the population at all, which loses all the meaning of using the VTGA. Besides the inefficiency of the first combination, we also find that when the virus population size goes up in this experiment, the VTGA requires more evaluations before finding the global optimum.

4.3 BEST CONFIGURATION

Table 11: Best Configuration of the VTGA for the Rastrigin's Function

Host Size	10
Virus Size	1000
Crossover Rate	0.7
Mutation Rate	0.01
Infection Rate	0.2

Spread Rate	0.2
Recovery Rate	0.8

With this setting, we run ten trials of the VTGA on the Rastrigin's Function of 5-dimension and 10-dimension, respectively. For the representation, we assign 100 genes for every dimension of the function. So the representations are of length 500 and 1000 respectively. The result is shown in Table 12.

Table 12: Average of the fitness value after 20000 evaluations for the Rastrigin's Function from **[9]**

Host Population Size	20	60	80	100	120	160
SSGA	3.714e-5	3.748e-5	4.466e-5	3.896e-5	3.725e-5	4.990e-5
VEGA	4.386e-5	3.704e-5	3.418e-5	**2.656e-5**	3.650e-5	1.511e-4

Table 11 is the result of an experiment on the 5-dimension Rastrigin's Function in **[9]**. In the experiment, the authors compared the results from the Steady State Genetic Algorithm (SSGA) and from the Virus Evolutionary Genetic Algorithm (VEGA). Table 11 show the average of fitness values after 20000 evaluations of 50 trials. In the experiment, the VEGA contains 10 viruses at all time. According to the result, the authors claim that when the virus population size of the VEGA is about 10% of the host population size, the algorithm performs the best. So we use this result as the standard for testing the VTGA.

Table 13: Result for the Rastrigin's Function

	10,000 Evaluations	20,000 Evaluations	100,000 Evaluations	Average Final Fitness	Evaluations Required before Meeting the Best
n =5	5.4295e-4	3.205933e-7	0	0	23400
n =10	3.0048000628	0.850642182	0.21820393	0.099949591	137900

In Table 12 we list the results from the VTGA for the Rastrigin's Function. We examine the VTGA on the 5-dimension and 10-dimension Rastrigin's Function. We use no-improvement-after-100-generations as the stopping criteria. We run 50 trials of the VTGA. We record the average fitness of the best found solution after 10000, 20000 and 100000 evaluations. And we also record the average best fitness found at the end of the VTGA and the average amount of evaluations are required for finding the best solution.

In [9], they don't give details about how they implement the VEGA. According to their description of the VEGA and our discussion in chapter 2, the fitness of a virus depends on the changes of fitness values of all hosts infected by that virus. Therefore, one evaluation of a virus required many evaluations of hosts infected by that virus. But since they don't describe their implementation in details, we don't know whether they include those extra evaluations or not when they count the evaluation amounts. But even so, based on Table 12, we see that the VTGA performs better on the average after 20000 evaluations than the VEGA and the SSGA. On the average, the VTGA just need 23400 evaluations, 17% more than 20000, to find the best solutions of each trial. What's more, the VTGA is able to find the global optimum each time, which shows the robustness of the algorithm.

In [9], the VEGA is tested on the Travelling Salesman Problem and the Knapsack Problem as well. But since the paper doesn't include detail settings of these problems, we are not able to repeat them and to compare the performance of the VEGA and the VTGA further.

4.4 FUNCTION OPTIMIZATION

In order to perform some quality tests of the VTGA, we run 50 trials of the algorithm to solve some function optimization problems. In the experiments, we use the same configuration acquired from the previous experiment for the Rastrigin's function. The results of 50 trials for every one of the six problems are listed in Table 14.

Table 14: Results for the Function Optimization Problems

Function	N	Global Best	20000 Evaluations		100000 Evaluations	
			Best	Average	Best	Average
De Jong's function	5	0	3.52E-17	1.6E-05	0	2.8E-22
Axis parallel hyper-ellipsoid	5	0	1.90E-18	9.8E-06	0	1.2E-22
Rotated hyper-ellipsoid	5	0	6.45E-14	0.00167	0	1.1E-20
Rosenbrock's valley	5	0	0.13	2.66363	0.12	2.54
Rastrigin's function	5	0	0	3.21E-07	0	0
Schwefel's function	5	-2094.91	-1787.40	-1451.82	-2094.71	-2010.84

From the results, we learn that the VTGA is able to obtain good results in most of the tests. It finds the global optimums of four problems and a result very close to the global optimum of the Schwefel's function. Although the result of the Rosenbrock's valley is not good compared to others, it is obtained by using the same configuration. Maybe the effectiveness could be improved by adjusting to another configuration. However, since we have proven that the algorithm is efficient enough to solve most of the cases well, we don't want to spend time on finding the perfect configuration for every function optimization problem.

4.5 TRAVELLING SALESMAN PROBLEM

To examine the robustness of the VTGA, we solve some TSP problems with the algorithm. In this thesis, we describe a solution of the TSP problems as a permutation of integers. And because the representation is different, we need to modify the operators of the VTGA accordingly to better match the TSP problems. There are many crossover and mutation operators that have been studied for the TSP problems [20]. But in this study, we only focus on the order based crossover

operator and the exchange mutation operator. The order based crossover was proposed by Syswerda [21]. It selects at random several positions in a parent tour, and the order of the cities in the selected positions of this parent is imposed on the other parent. The original based order crossover operator generates two offspring by exchange positions of two parents. In the VTGA, we modify the order based crossover operator to fit the feature of the algorithm. The revised order based crossover operator works like this: when we have a pair of two virus solutions, we compare their fitness values to determine which one is better. For example, we have two viruses

V1: (1 2 3 4 5 6 7 8) and V2: (2 4 6 8 7 5 3 1)

And V1 is better than V2. Then we randomly select a number p between 0 and the Crossover_Rate, which is smaller than or equals to 1. Then $\lfloor p * GENE\_SIZE \rfloor$ genes will be selected and copied from the stronger virus to the weaker virus following the order of the cities in the selected positions. Let's assume $\lfloor p * GENE\_SIZE \rfloor$ = 3. We randomly pick a subtour length 3 from V1 and copy the rest from the V1 to the V2. The V2 becomes

V2: (1 2 3 * * 7 8)

Finally, we replace the resting cities to the weaker virus in the same order in which they appear in the weaker virus. The resulting V2 is

V2: (1 2 3 4 6 5 7 8)

Since we only allow information to be copied from good virus to bad virus in the VTGA, so V1 will be the same after the crossover. We make similar modification to the infection operator as well so that information will be copied from a host to its virus when infection happens.

The exchange mutation operator is simple. The VTGA randomly selects two cities in a solution and exchanges them. After all the modifications are done, we run 30 trials of 5 different TSP problems. In Table15 are the results of the experiments.

Table 15: Results for the Travelling Salesman Problems

TSP Instance	HS	VS	CR	MR	IR	SR	RR	Global Optimum	Best	Average	STD	Average number of evaluations (Million)
Bays29	200	1000	0.3	0.01	0.7	1.0	0.8	2020	2020	2020.2	1.08	0.34
Swiss42	200	1000	0.3	0.01	0.7	1.0	0.8	1273	1273	1274.7	5.92	2.5
Berlin52	200	1000	0.3	0.01	0.7	1.0	0.8	7542	7542	7570.3	56.89	7.1
Eil76	200	1000	0.3	0.01	0.9	1.0	0.5	538	538	548.9	5.62	10.8
Ch150	200	1000	0.1	0.01	1.0	1.0	0.8	6528	7107	7803.7	367.56	14.7

Considering the difference of the TSP problems and the impact of the difference to the configuration, we use a slightly different setting of parameters when performing the experiments. Because our purpose in this experiment is to find out if the VTGA remains effective when it solves other problems with a different representation. From the results, we learn that the algorithm is able to find the global optimum in four problems and achieve a very good average fitness as well. Although the algorithm is not able to find the global optimum of the Ch150, in general, we believe the VTGA is an effective algorithm for solving the TSP problems.

4.6 FOREST PLANNING PROBLEM

In this section, we apply the VTGA to the 73-stand forest planning problem. To measure the effectiveness of the algorithm, we implement four Genetic Algorithms, two of which use 2-point crossover and the other two use uniform crossover. In the group of GAs using 2-point crossover, one of the two has the elitism operator, which is a method that makes sure the best solution found is in the population of every generation. And the other GA in the same group doesn't have this operator. Meanwhile, only one GA of the uniform crossover has the elitism operator as well. The reason for creating different GAs like that is because we are not sure which crossover implementation is better for the FP problem and if elitism is a good mechanism for the FP problem. The configurations of four GAs are listed in Table 5. The Forest Planning problem is a constrained optimization problem. A valid solution of the problem has to obey all rules listed in

the problem. To acquire valid solutions using a search algorithm, there are two common methods. One method is to include a penalty function in the objective function. When a solution has violations of rules, the penalty function gives punishment to its fitness, which is usually an increment for a minimization problem and a decrement for a maximization problem. To optimize the fitness of the solution, the search algorithm needs to reduce the punishment by minimizing the amount of violations. The advantage of this method is that it is easy to implement. We just need to modify the objective function so that it could calculate the amount of violations and the corresponding punishment. But the disadvantage of the method is that it requires extra work to figure out the appropriate impact of the punishment to the fitness value. If the impact is too small, it is possible for an algorithm to return a solution still with violations because the punishment for violations is negligible. But on the other hand, if the effect is too big, an algorithm will force every solution it generates to obey all constraints. In that case, many effective patterns will be eliminated quickly from the population at the early stage of search because the solutions containing these patterns have violations. The second method is to repair invalid solutions so that they become violation-free. To implement this process, the repair function has to go through all constraints for every solution. If a solution has any violations, the repair function will assign some new values to some positions of the solution to make sure that all existing violations are eliminated and no new violation is created. But this method is not always available. For some problems with strict constraints, only a few solutions are valid. Therefore, finding these solutions is as difficult as finding the optimum. In that case, the effort to repair a solution is possibly very intense. But in our study, the FP problem is not one of those problems. While fixing a harvest schedule, the repair function could always assign the no-cut option to any stands without causing

new violations. Since it is relatively easy to implement a repair function for the FP problem, therefore, we use this method to make sure every solution found by the VTGA is valid.

Table 16: Configuration for four GAs

	GA1	GA2	GA3	GA4
Population	1000	1000	1000	1000
Selection	Tournament	Tournament	Tournament	Tournament
Crossover	2-point	2-point	Uniform	Uniform
Crossover Probability	80%	80%	80%	80%
Mutation	Random Resetting	Random Resetting	Random Resetting	Random Resetting
Mutation Probability	40%	40%	40%	40%
Mutation Rate	1%	1%	1%	1%
Elitism	Yes	No	Yes	No
Repair	Yes	Yes	Yes	Yes
Stop Criteria	5,000,000 Evaluations	5,000,000 Evaluations	5,000,000 Evaluations	5,000,000 Evaluations

In order to find the best configuration of the VTGA for the FP problem, we go through the same process of finding the best configuration for the Rastrigin's function. And it turns out the best configuration for the FP problem (Table 14) is quite different from that for the Rastrigin's function.

Table 17: Configuration of the VTGA for the FP problem

Host Size	10
Virus Size	1000
Crossover Rate	0.3
Mutation Rate	0.01
Infection Rate	0.9
Spread Rate	0.8
Recovery Rate	0.8

Table 18: Comparison between the VTGA and four traditional GAs

	1,000,000 Evaluations	2,500,000 Evaluation	5,000,000 Evaluations
VTGA	10436823.65	8831595.131	7618037.40
GA1	7868882.89	7208875.41	7039678.77
GA2	12799144.3	12011354.9	10509188.3
GA3	10544942.7	9677261.27	9204251.9
GA4	8271758.47	8007286.16	7984191.05

To compare the performance of the VTGA and four different GAs, we run 30 trials of each of them. We record the fitness of the best found solutions after 1000000, 2500000 and 5000000 evaluations. Table 15 lists the average of our records. From the table, we know that the

best GA for the FP problem is GA1, which uses 2-point crossover and elitism. The VTGA performs better than GA2, GA3 and GA4. But the VTGA is not able to find better solutions than that found by GA1 within 5000000 evaluations. So we know that GA1 converges to good solutions faster than the VTGA. But this is what we expect from the comparison because we already know that the Genetic Algorithm has the ability to converge quickly. Like we discuss in the review, this is a problem of the GA, because as it converges quickly, it loses diversity quickly. Effective patterns are eliminated from the population soon, which prevents the GA from improving its solutions even more. So when we examine the VTGA, we tend to compare its performance with the GA's performance with at least 50 million evaluations.

By the previous experiment, we prove that the GA1 is the best one of the four GAs, so we are going to run it as well as the VTGA again for 30 trials with 50 million evaluations as the stopping criteria.

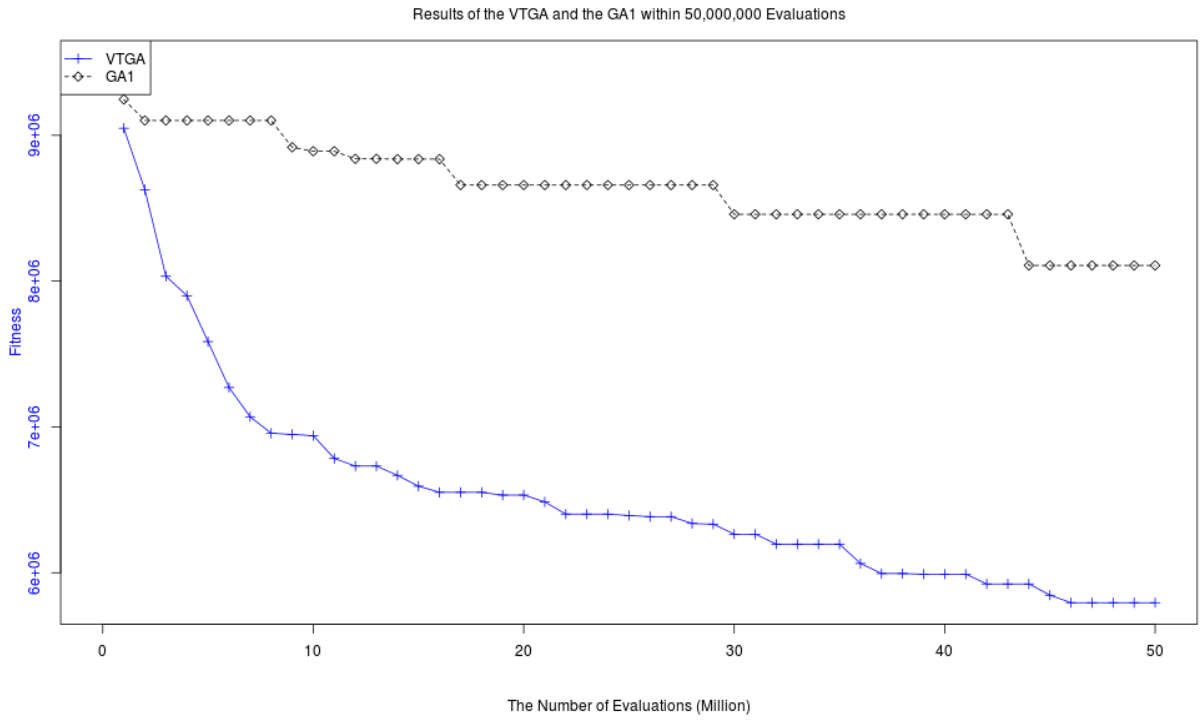


Figure 21: Results of the VTGA and the GA1 within 50,000,000 Evaluations

Figure 21 shows the results of the VTGA and the GA1. The fitness values in the graph are the average of 30 trials. After 1 million evaluations, it is difficult for the GA1 to improve the solutions it finds. The best solution found by the GA1 remains the same for many generations before any improvement. On the other hand, the VTGA improves its solutions more smoothly and effectively. After every 1,000,000 evaluations, the algorithm is able to improve its best solution more or less. Table 16 shows more details about the results. Over 30 trials, the VTGA outperforms the GA1 on the average fitness and the fitness of the best found solution. And the VTGA finds the global optimum 21 times out of 30, which means about 69.2301% accuracy of finding the global optimum.

Table 19: Comparison between the VTGA and the GA1 in detail

Algorithm	Average Fitness	The Best Fitness Found	The Number of Trials Finding the Global Optimum
VTGA	5794786.28011043	5500330.279304971	21
GA1	8107453.58754281	5502420.092225004	0

To assess the effectiveness of different heuristics for solving forest planning problems, three hypothetical forests are used in [19]. Among them, the 625-unit hypothetical southern U.S. forest is the most complicated one. It was designed as a 25 by 25 unit grid, with each grid cell representing 10 ha. The time horizon of this example is assumed to be 15 years, hence there are three time periods, every of which is 5 years long.

To solve the 625-stand FP problem, we continue to use the parameter configuration for solving the 73-stand FP problem. Because we don't have an estimation of how many evaluations are required approximately for finding good solutions, so we use no-improvement-in-1000-generations as the stopping criteria. Table 17 shows the summary of 10 trials.

Table 20: Result for the 625-stand FP

Best Solution	Average Solution	Average of Evaluation Times
109,596,669,572	117,644,347,372	27,971,133

We notice that the average of evaluation times is just 27,971,133, which is relatively small compared to the number of evaluations we use for the 73-stand forest. So we change the stopping criteria and make the VTGA stop after 60,000,000 evaluations. As a result, the VTGA is able to reach 97,232,163,832. And since we consider with more genes in representation, it may require more host population to record effective patterns of the 625-stand FP problem. Therefore, we make another attempt, in which we increase the host population size to 100 and change the stopping criteria back to no-improvement-in-1000-generations. The result shows that with this configuration, the VTGA is able to reach 83,509,683,812. Besides, in the same trail, the VTGA is able to find 100 solutions, the average fitness of which is 94,885,088,126.

CHAPTER 5

CONCLUSION AND FURTHER WORKS

In this thesis, we present a new search method, Virus Transmission Genetic Algorithm. We have proven that this algorithm is effective in solving some function optimization problems, some TSP problems, 73-stand and 625-stand FP problems. And through experiments, exciting features of the VTGA are revealed.

Summary

Table 21: Summary of the Experiment Result

Algorithm	The 5-dimension Rastrigin Function		73-stand FP Problem		625-stand FP Problem	
	Average	Best	Average	Best	Average	Best
GA	-	-	8107453	5,502,420	-	-
SSGA	0.00003714	-	-	-	-	-
VEGA	0.00002656	-	-	-	-	-
VTGA	0	0	5,794,786	5,500,330	-	83,509,683,812
$RWPSO_{f2}$	-	-	7,492,459	5,500,330	150,819,800,640	118,239,623,212
$RWPSO_{f3}$	-	-	5,844,508	5,500,330	100,862,133,880	91,224,899,372
$RWPSO_{f3\text{-}pb}$	-	-	5,786,583	5,500,330	95,872,673,094	87,444,889,432

In Table 18, we summarize the results acquired from our experiments. The Rastrigin Function is a relatively simple problem. By experiments, we have proven that the VTGA is one competitive and efficient search algorithm, compared with the steady-state GA (SSGA) and the Virus-Evolutionary GA (VEGA) in function optimization. Then we move on to two more complex constrained optimization problems, the 73-stand and the 625-stand forest planning problems. We prove in our study that the VTGA can outperform the generational GA on the 73-stand FP problem. With the same number of evaluations, the VTGA performs better on either the average fitness or the best fitness. For comparison, we include some external results from [22] in the

discussion. In [22], the multi-valued algorithm Roulette Wheel PSO (RWPSO), guided by different functions f_2, f_3 and f_{3-pb} [22], is proposed for the forest planning problems. For the 73-stand forest, the VTGA is able to find the global optimal solution of 5,500,330 as well as three RWPSOs. And the VTGA outperforms two RWPSOs on the average and is slightly worse than the third. However, since the RWPSO is formulated specifically for multi-valued nominal variable problems and the VTGA is designed more generally without including beneficial heuristics for the FP problems, the VTGA requires much more evaluations, which leads to the fact that we are not able to run enough trials of the 625-stand forest to compare the average fitness with the RWPSOs. But even so, we are able to reach a better 625-stand schedule than all the RWPSOs in a few trials.

Favoring Bad Solutions

In the population, traditional genetic algorithms favor good solutions by giving more opportunities to outstanding individuals for anticipating recombination. But due to the nature of optimization problems, improvement of good solutions is harder and harder while their fitness values increase. In the VTGA, on the other hand, through a mechanism simulating a virus attack, weak hosts are infected more often, which leads to a fact that more attention is placed on improving bad solutions.

Isolated Evolution

In the VTGA, viruses are isolated within different hosts. The idea is similar with that of the isolated subpopulation of the Island GA. Evolution, therefore, involves only viruses in the same host. Through experiments, we find that this mechanism helps with maintaining diversity in the population.

Different Implementation of Operators

In this study, we implement a one-way crossover operator sending genetic information only from strong individuals to weak individuals. And the way this operator delivers information is almost the same as the way uniform crossover performs in a traditional GA, which has various ways to implement crossover. We believe there are other ways to implement crossover and other operators in the VTGA.

The role of the algorithm

Based on our study, we find that the VTGA is able to solve the premature convergence problem efficiently. In our experiment, the GA converges quickly at the early stage of a computation. But after it converges, it is difficult of the algorithm to escape the local optimum. Generally, once the GA is trapped in a local optimum, it takes the GA many generations to further improve the solutions of the population. On the other hand, because the VTGA uses the host population to record good solutions of different areas of the search space, it preserves diversity well. Through experiments, we have shown that the VTGA improves solutions more smoothly than the GA.

Besides, when one computation of the GA finishes, generally the best solution found by the GA dominates the whole population, which means most of the individuals in the population are copies of the best solution. Therefore, the GA can only return one solution after it converges. However, the VTGA is able to give users many excellent solutions when it finishes one computation. In Table 17, we show the solutions recorded by a size 10 host population at the end of one computation of the VTGA. We can see that in the host population, there is not only the global optimal solution but also other solutions which have relatively good fitness values. This feature of the VTGA can help us understand the search space of a problem better by giving us representative solutions from different areas.

Table 22: Solutions Stored in the Host Population

Solution	Fitness
121223223311223110022303233113132323121332011122223323221331233333333211 2	7030243.24052599
121223213311223110022303223213132332121233011122223323221131231233333221 2	6578168.329585003
121223223311223110022333223113132331121203311122223323231132211333323111 1	6213043.9777689725
121223223311223110022303223113132332121233011122223323221331233333323111 2	6444498.948634986
321232223311232310012323232112123232122302311111233323331222213333333111 2	10935572.966882974
121223223311223110022303223113132332121233011122223323221331233333323111 2	6444498.948634986
121223223311221303022313223213332332111203211122223323231232213333323311 2	5500330.279304971
313232123311212333012323222113333232111203212211213323132131221333333112 1	12899613.106605
121213132312223121022303223113132331121233011112213323221333213233332112 1	7773060.480105973
121223132311221303022313233113332331112302211112223323231131223333322112 1	7407795.865192981

Solving other Problems

In this thesis, we apply the VTGA to the Rastrigin's function, the 73-stand FP problem and the 625-stand FP problem. But for further discussion, we are looking forward to solving more problems with the VTGA.

REFERENCES

[1] J. H. Holland, Adaptation in natural and artificial systems, Ann Arbor, MI: University of Michigan Press, 1975.

[2] L. D. Davis, Handbook of Genetic Algorithms, New York, NY: Van Nostrand Reinhold, 1991.

[3] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Reading, MA: Addison-Wesley, 1989.

[4] J. P. Cohoon, S. U. Hegde, W. N. Martin and D. S. Richards, "Punctuated equilibria: a parallel genetic algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, Hillsdale,NJ, 1987.

[5] R. Lohmann, "Application of Evolution Strategy in Parallel Populations," in *PPSN I Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, London, UK, 1991.

[6] W. N. Martin, J. Lienig and J. P. Cohoon, "Island (migration) models: evolutionary algorithms based on punctuated equilibria," in *Evolutionary Computation 2: Advanced Algorithms and Operators*, London, UK, IOP Publishing, 200, pp. 101-124.

[7] M. Ridley, Evolution, 2nd edition, Wiley-Blackwell, 1996.

[8] N. Kubota, T. Fukuda and K. Shimojima, "Virus-evolutionary gentic algorithm for a self-organizaing manufacturing system," *Computers & industrial engineering*, vol. 30, no. 4, pp. 1015-1026, 1996.

[9] N. Kubota, K. Shimojima and T. Fukuda, "The role of virus infection in virus-evolutionary genetic algorithm," in *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996.

[10] L. N. de Castro and J. Timmis, Artificial Immune Systems: A New Computational Intelligence Approach, London, UK: Springer-Verlag, 1996.

[11] P. Ziegler, The Black Death, New York, NY: Harper & Row, 1969.

[12] M. Oldstone, Viruses, Plagues and History, New York, NY: Oxford University Press, 2000.

[13] G. Min-Oo, and P. Gros, "Erythrocyte variants and the nature of their malaria protective effect," *Cellular Microbiology*, vol. 7, no. 6, pp. 753-763, 2005.

[14] S. Tishkoff and B. Verrelli, "Patterns of human genetic diversity: implications for human evolutionary history and disease," *Annual Review of Genomics and Human Genetics*, vol. 4, pp. 293-340, 2003.

[15] J. C. Stephens, D. E. Reich, D. B. Glodstein, H. D. Shin, M. W. Smit, M. Carrington and C. Winkler, "Dating the origin of the CCR5-Delta32 AIDS-resistance allele by the coalescence of haplotypes," *American Journal of Human Genetics*, vol. 62, pp. 1507-1515, 1998.

[16] A. P. Galvani and M. Slatkin, "Evaluating plague and smallpox as historical selective pressures for the CCR5-Delta 32 HIV-resistance allele," *Proceedings of the National Academy of Sciences USA*, vol. 100, no. 25, pp. 15276-15279, 2003.

[17] R. M. Anderson and R. M. May, "Population biology of infectious diseases: Part 1," *Nature*, vol. 280, pp. 361-367, 19979.

[18] M. Molga and C. Smutnicki, "Test functions for optimization needs," 2005. [Online]. Available: http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf.

[19] P. Bettinger and J. Zhu, "A new heuristic for solving spatially constrained forest planning problems based on mitigation of infeasibilities radiating outward from a forced choice," *Silva Fennica*, vol. 40, no. 2, pp. 315-333, 2006.

[20] P. Larra ñaga, C. M. Kuijpers, R. H. Murga, I. Inza and S. Dizdarevic, "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 129-170, 1999.

[21] G. Syswerda, "Schedulel Optimization Using Genetic Algorithms," in *Handbook of Genetic Algorithms*, New York, NY: Van Nostrand Reinhold, 1991, pp. 332-349.

[22] J. Smythe, W. D. Potter and P. Bettinger, "Application of a new multi-valued particle swarm optimization to forest harvest schedule optimization," in *Proceedings of the 2012 International Conference on Genetic and Evolutionary Methods*.