Mapping physical topology with logical topology using genetic algorithm.

Zhiyong Michael. Liu
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/2663
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®
Mapping Physical Topology with Logical Topology Using Genetic Algorithm

By
Zhiyong Michael Liu

A Thesis
Submitted by the Faculty of Graduate Studies and Research through the school of Computer Science in Partial Fulfillment of the Requirement for the Degree of Master of Science at the University of Windsor

Windsor, Ontario, Canada
2001

© 2001 Zhiyong Michael Liu
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-62245-2
Abstract

With the rapid development of Internet and the increasing demand for broad bandwidth, optical network becomes more and more popular due to its high speed and huge capacity. An optical fiber can carry several light wavelengths. By assigning these wavelengths to the transmitters/receivers of optical hops, the logical topology for a physical network can be defined. The main advantage of such architecture is its reconfigurability. However, determining the best wavelength assignment and flow assignment becomes an issue. Existing approaches addressing these problems, based on optimization techniques, are unsatisfactory with large networks because of the large number of constraints.

By specifically appointing a logical topology with good properties, the exponential constraints can be avoided and a good solution can be found with much less effort. Based on this idea, a genetic algorithm is proposed to solve the problem. The objective of this study is to investigate genetic algorithm's suitability for this logical design problem. To calculate the congestion, the Alternative Path Algorithm is introduced as the objective function. New crossover strategies like Sub–graph Crossover, Cluster Crossover and Random Crossover are introduced. Different test schemes are evaluated in this thesis.

The good–gene effect, chromosome contention, and dropping coverage ratio/protection ratio phenomenon raise lots of challenges in practice. But over 90% chance of getting the optimal solution for 9–node networks makes this algorithm very attractive for further study.
Dedication

This work is dedicated

To my grandfather, whom I never have a chance to know;

To my grandmother, whom I never have a chance to say goodbye;

To my parents, Yuxiang and Peixia, who through personal sacrifices permitted me to complete my Master thesis successfully, from whom I inherited determination and the spirit of never give up;

Also to my beautiful niece, whose cute face is my forever driving force.
Acknowledgments

I would like to acknowledge my sincere gratitude towards my supervisor Dr. A. Jaekel: for her valuable advice, which is always available when I need it; for her confidence on me, which left me lots of room for imagination and self-development; and for her time commitments, without which this work would never be achieved. It is a privilege to study and work under her supervision.

I would also like to thank Dr. S. Bandyopadhyay for his guidance; Dr. A. Ngom for his expertise in genetic algorithms; and Dr. N. Biswas for his suggestions to improve this thesis.
# Table of Contents

Abstract ........................................................................................................ iii

Dedication ....................................................................................................... iv

Acknowledgments ........................................................................................ v

List of Figures ............................................................................................ ix

List of Tables .............................................................................................. x

1. The Configuration Problems in Optical Network Design ...... 1
   1.1 Problem Outline ................................................................................. 2
   1.2 Existing Solutions ............................................................................. 3
   1.3 The Proposed Approach ................................................................... 5
   1.4 Thesis Objective ................................................................................ 7
   1.5 Thesis Structure ............................................................................... 8

2. Background Introduction .................................................................. 9
   2.1 Introduction to De Bruijn Graph ...................................................... 9
   2.2 Overview of Genetic Algorithms .................................................... 11
      2.2.1 Simple Genetic Algorithms and its Variations ...................... 13
      2.2.2 The Power of Genetic Algorithms ...................................... 17
         2.2.2.1 Data Representation Abstraction ................................ 17
         2.2.2.2 Parallel Search ............................................................... 19
         2.2.2.3 Black Box Search ......................................................... 20
         2.2.2.4 Probabilistic Rule ......................................................... 20
      2.2.3 Transformation ........................................................................... 22
2.2.4 Complement ........................................................................... 24

3. New Approach: A GA for the Mapping Problem .............. 26
   3.1 Chromosome Representation ............................................ 27
   3.2 Initialization ..................................................................... 28
      3.2.1 Alternate Path Algorithm ......................................... 29
   3.3 Selection, Crossover and Mutation ................................. 31
   3.4 Objective Function and Fitness Value ................................. 33
      3.4.1 Fitness Value .......................................................... 37
   3.5 Stop Criteria .................................................................... 38
   3.6 Conclusion .................................................................... 40

4. Crossover Strategies ................................................................. 41
   4.1 Order Crossover ............................................................... 41
   4.2 Cluster Crossover ............................................................. 43
   4.3 Sub-graph Crossover ......................................................... 44
   4.4 Random Crossover .......................................................... 46
   4.5 Mixed Crossover .............................................................. 48
   4.6 Complement Operation .................................................... 48
   4.7 Conclusion .................................................................... 50

5. Experimental Results and Performance Analysis .............. 51
   5.1 Test Environment and Control Parameters ....................... 51
   5.2 Results for HC, ES and Order Crossover ......................... 55
   5.3 Sub-graph and Cluster Crossover Strategies ...................... 58
   5.4 Random and Mixed Crossover .......................................... 60
   5.5 Inserting Good Genes into Initial Population ...................... 62
   5.6 Conclusion .................................................................... 64

6. Conclusions and Future Work ................................................. 67
   6.1 Conclusions .................................................................... 67
6.1 Future Work ........................................................................................................ 69

References ............................................................................................................. 71

Appendix A Definitions ......................................................................................... 78

Appendix B Abbreviations ..................................................................................... 81

Appendix C Traditional Search Methods and Their Limitations .......... 82

Appendix D Advanced Topics on GA ................................................................. 85

Vita Auctoris .......................................................................................................... 92
List of Figures

Figure 1–1. Physical Topology ......................................................... 1
Figure 1–2. Logical Topology .......................................................... 1
Figure 1–3. A Mapping in a De Bruijn Graph (2^3) ............................... 6
Figure 2–1. A 2^3 De Bruijn Graph .................................................. 10
Figure 2–2. An Abstract View of Genetic Algorithm ............................. 12
Figure 2–3. The Generation Procedure .......................................... 12
Figure 2–4. Comparison: A General Perspective ............................... 21
Figure 3–1. The Proposed Genetic Algorithm .................................... 27
Figure 3–2. Alternative Path Algorithm .......................................... 30
Figure 3–3. Outline of Objective Function ....................................... 33
Figure 3–4. De Bruijn Graph for Objective Function ............................ 35
Figure 3–5. De Bruijn Graph after Processing t_{AB} ............................... 36
Figure 3–6. De Bruijn Graph after Processing t_{AD} ............................... 37
Figure 3–7. Stop Criteria ............................................................... 39
Figure 4–1. An Example of Cluster Centered at 011 ............................ 43
Figure 4–2. The Mask Operation .................................................... 47
Figure 5–1. An Optimal Mapping for Traffic Matrix in Table 5–1 .......... 53
List of Tables

Table 1–1. A Traffic Matrix ................................................................. 6
Table 3–1. Traffic Matrix for Example 3–1 ........................................... 34
Table 4–1. Chromosome C₁ ................................................................. 41
Table 4–2. Chromosome C₂ ................................................................. 42
Table 5–1. An Artificial Traffic Matrix for 3² De Bruijn Graph .......... 53
Table 5–2. Results Using ES/HC .......................................................... 55
Table 5–3. Results Using OCX ............................................................. 56
Table 5–4. Test Result for SCX ............................................................ 58
Table 5–5. Test Result for CCX ............................................................ 59
Table 5–6. Test Result for RCX ............................................................ 60
Table 5–7. Test Result for MCX ............................................................ 61
Table 5–8. Test Result for Good-gene Effect ....................................... 63
Table 5–9. Summary for Test Results ................................................. 66
CHAPTER 1

The Configuration Problems in Optical Network Design

Optical fibers are the major telecommunication medium today in local and metropolitan area networks because of their large bandwidth, high speed and low loss. Wavelength Division Multiplexing (WDM) allows a number of signals to be combined and transmitted over a single fiber [Aca93]. By assigning these wavelengths to transmitters/receivers of optical hops, different lightpaths can be created, which are end-to-end communication paths passing through one or more fibers and using one WDM channel per fiber [LA91]. The logical topology of an optical network is a graph where each node (often called end-node) represents a computer in the network and an edge between two end-nodes $u$ and $v$ represents the fact that there is a lightpath between nodes $u$ and $v$. Figure 1–1 is an example of a physical topology where A is physically connected to B, B to C, C to D and D to A. Figure 1–2 is the logical topology for this physical network based on the wavelength assignment in Figure 1–1.

![Figure 1–1 Physical Topology](image1)

![Figure 1–2 Logical Topology](image2)
Wavelength routed optical networks (WRONs) are the new generation of optical
networks which route optical signals selectively based on the wavelength, \( \lambda \), and the
input port. Such networks are reliable, highly utilized and logically reconfigurable
[LA91]. This chapter reviews the general problems in lightwave routed optical
network design, the existing solutions to these problems and their limitations. Then it
outlines the proposed approach and discusses the thesis objective.

1.1 Problem Outline

Optical networks can be classified into two categories – single hop and multihop
[Muk97]. In a single hop network, a signal transmitted from a source node reaches
the destination node directly without going through any other end-nodes. In a
multihop network, information from a source is routed through several intermediate
end-nodes before reaching the destination. This architecture was introduced by
Acampora [Aca87], in order to address the scalability problem of purely single hop
networks. Multihop networks are generally used in situations where it is not feasible
to have a lightpath between every source-destination pair. As shown in Figure 1-1,
the physical topology has little relationship to the logical topology and, by changing
the lightpaths, the same physical topology can support a very large number of logical
topologies. These different logical topologies have different performance
characteristics and an important research area is to determine the optimal logical
topology for a given optical network with a specified physical topology, expected
traffic between all node pairs and other physical characteristics. For optical networks,
the two factors that have been used to evaluate the performance of a multihop
network are its delay and its congestion [RS98] where the delay is the time taken by a
signal to travel from the source node to the destination node and the congestion is the
maximum load offered to any logical link. In the context of Wide Area Networks (WAN), the delay is determined by the propagation delay and the primary concern is to reduce the congestion of the network [RS98] which reduces the cost of the optical hardware.

1.2 Existing Solutions

In recent years, there has been considerable interest in the area of logical topology design for multihop networks. The logical topology, to be designed, is represented by the directed graph $G(N, A_t)$, where $N$ is the set of nodes and $A_t$ is the set of lightpaths established. The problem essentially consists of two parts:

i) connectivity problem – to determine the set of lightpaths

ii) flow assignment problem – to determine an optimal routing over the logical topology.

Two main approaches have been used to solve this problem – using mathematical optimization techniques and using regular topologies. The first approach is based on the traffic demands of the various node pairs and attempts to design an optimal logical topology and determine a routing scheme over the given physical topology. The combined topology design and routing problem can be formulated as a mixed integer linear program (MILP) which minimizes the congestion of the network, subject to an upper bound on the maximum (propagation) delay for any communication [RS96]. The resulting logical topology is generally an arbitrary graph, with no regular structure. The main drawback of this approach is that the number of constraints, $O(N^3)$, grows rapidly with the size of the network and the approach becomes infeasible even for moderate sized networks. The problem can be simplified by
decoupling the logical topology design from the problem of determining the optimal routing scheme [RS96] between all source destination pairs. Heuristics have been proposed for logical topology design and the problem of finding an optimal routing between different nodes using the logical topology can be formulated as a linear programming (LP) problem. Even the LP formulation generates \(O(N^3)\) constraints and is infeasible for moderate sized networks.

The second approach that has been proposed for logical topology design is to ignore the traffic demands and to design regular logical topologies that have some desirable and well known graph theoretical properties such as predetermined routing, low diameter \(O(\log N)\) for many regular topologies where \(N\) is the number of nodes in the network) and high connectivity. Many topologies including the torus [MBLN93], the Perfect Shuffle [HK99], the Hypercube [Dow92], the De Bruijn graph [SR91] and others have been proposed for optical networks. The regular topology based designs were originally proposed for broadcast and select networks and have not been investigated thoroughly for wavelength routed networks. Many researchers feel that such topologies may be appropriate only if the traffic is uniform or when the traffic pattern is not known and that their usefulness in wide-area, wavelength routed networks is questionable [RS98].

There are also a number of other strategies for logical topology design. The most recent attempt by Gazen and Ersoy uses a Genetic Algorithm (GA) to solve the connectivity problem [GE99]. They introduce Minimal Hop with Flow Deviation as their objective function. But again, they formulate the flow assignment problem as linear equations. Although it shows promising future for large size problems, its result is only "comparable" with those of heuristic algorithms. Other algorithms also include
Ersoy and Panwar’s simulated annealing algorithm [EP93], Zhang and Acampora’s heuristic wavelength assignment algorithm [ZA95], and Ramaswami and Sivarajan’s design approach for maximizing the traffic load on the logical topology [RS95] etc.

1.3 The Proposed Approach

The approach proposed here uses a regular graph, the De Bruijn graph, as the logical topology. The rationale for using the De Bruijn graph is that it has a low diameter, a large connectivity and a simple routing scheme. In the De Bruijn graph, it is known that there are a number of node disjoint paths from a source to a destination, each having almost the same number of nodes as in the shortest path route [SSB88]. This means that the routing scheme can make full use of the rich interconnections available in a regular graph and the fact that numerous, relatively short, routes exist between any source–destination. By specifying the target logical topology, the original topology design problem is reduced to a much simpler one of finding a good mapping between the logical and physical nodes. Thus, a good solution can be expected to be found with much less effort.

The target problem is presented as follows: given the traffic matrix, find out the proper mapping and flow assignment in a De Bruijn graph such that the congestion is minimized. For example, Table 1–1 is a traffic matrix and Figure 1–3 is a De Bruijn graph. What needs to be done is to find out how to map physical nodes (A, B, ..., H) to logical nodes (000, 001, ..., 111) in the De Bruijn graph. In this example, A is mapped to 001, B to 000, C to 010 etc. The question is that can such a mapping minimize the congestion? The basic idea is that if two nodes have very heavy traffic, they should be assigned to be as close to each other as possible. And this is the
heuristic used in this research. Genetic Algorithms (GAs) have been decided to be used to solve the above problem.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1-1. A Traffic Matrix

Figure 1-3. A Mapping in a De Bruijn Graph ($2^3$)

This research only considers the traffic matrix, which provides the information about the data flow distribution. Other physical constraints are ignored. It is assumed the same capacity for each link and also assumed that there are enough channels per fiber to accommodate all the lightpaths.
This research tries to find a mapping with reasonable (rather than minimal) congestion. The word "reasonable" is used instead of "minimal" because:

1. In most cases the minimal congestion is unknown;
2. Even if the minimum congestion is known, it may be infeasible to find it for larger networks.

1.4 Thesis Objective

This thesis uses genetic algorithms to find a mapping between the logical and physical nodes provided that the traffic matrix is given. The objective of this thesis is to test whether GA is suitable for this type of problem. In order to accomplish this, the following questions have to be answered:

1. What are the control parameters for the proposed algorithm? What is the proper population size? How many generations are required before the algorithm stops? What are the objective function and crossover strategy?
2. How well does the GA perform? How close is the result to the global optimum? How much time does it take to achieve such result?
3. What are the important factors that affect the performance? How do they contribute to the success or cause the failure?

This research investigates if GA can produce reasonable solutions and determines the network size that the proposed approach can handle. Although, it is beyond the scope of this thesis, it is the ultimate goal of this research to find out how logical topologies based on regular graphs compare with the arbitrary topologies obtained by using optimization techniques.
1.5 Thesis Structure

This thesis is organized as follows. Chapter 2 reviews some background information including De Brujin graphs and genetic algorithms. Chapter 3 proposes a new approach for the optical network design problem and Chapter 4 introduces several new crossover strategies for the proposed GA. The experimental results are presented and analyzed in Chapter 5. Chapter 6 addresses the contributions of this thesis, suggests directions for future work and concludes with a critical summary.
CHAPTER 2

Background Introduction

This chapter reviews some background information that is relevant to the remainder of this thesis. In particular, the structure and important properties of the De Bruijin graph are discussed, which has been selected as the target logical topology. It also reviews GA and discusses why GA is an appropriate tool for this particular application. Details of more advanced issues related to GA like parallelism, schemata, and epistasis are given in Appendix D.

2.1 Introduction to De Bruijn Graph

A De Bruijn graph [SR91], denoted as $B(d, k)$, is a regular directed graph, of degree $d$ and diameter $k$, with $d^k$ nodes. Each node with a logical node number $n_i$ has a label $L_i$, associated with it. The label is represented by a string consisting of $k$ elements, i.e.,

$$L_i = u_0 u_1 \ldots u_{k-1}, \text{ where } u_i \in \{0 \ldots d-1\} \text{ and } n_i = \sum u_i \cdot d^{i-1}.$$ 

Two nodes $u, v$ will have an edge if and only if $v$ is a left-shifted version of $u$, that is, if $u = u_0 u_1 \ldots u_{k-1}$, then $u \rightarrow v$ exists iff $v = u_1 u_2 \ldots u_{k-1} i$, for any $i \in \{0 \ldots d-1\}$. There are $d$ nodes in a De Bruijin graph, which have self loops. These nodes are represented by strings, which have all symbols the same.

The successor of a node $u$ in the logical topology is any node $v$ such that an $u \rightarrow v$
exists in the topology. For a De Bruijn graph, the set $S(u)$ of all successors of a node $u$ is given by $S(u) = \{v \mid v = u_0 \ldots u_{k-1}, i \in (0 \ldots d-l)\}$. Similarly, the predecessor of a node $u$ in the logical topology is any node $v$ such that $v \rightarrow u$ exists in the topology.

For a De Bruijn graph, the set $P(u)$ of all predecessors of a node $u$ is given by $P(u) = \{v \mid v = u_0 \ldots u_{k-2}, i \in (0 \ldots d-l)\}$. A node $v$ is adjacent to a node $u$ in the logical topology iff $v$ is a successor or a predecessor of $u$. The set of all adjacent members of node $u$ in the logical topology is given by $\text{ADJ}(u) = S(u) \cup P(u)$.

Figure 2–1 shows a $2^1$ De Bruijn graph. Each node is a left-shifted version of its predecessors; and each predecessor is a right-shifted version of its successors. The De Bruijn graph has been proposed in the literature as the potential candidate for logical topology design. This is mainly because of its simple routing strategy and rich interconnections. Research has shown that De Bruijn graphs, as logical topologies, are better able to handle large size networks than ShuffleNets [SR91].

![Figure 2–1. A $2^1$ De Bruijn Graph](image-url)
One of the most important properties of a De Bruijn graph is its low diameter. Consider two nodes $u = u_0 u_1 \ldots u_{k-1}$, $v = v_0 v_1 \ldots v_{k-1}$. There is always a walk from $u$ to $v$ of length $k$ and it is given by

$$u = u_0 u_1 \ldots u_{k-1} \rightarrow u_1u_2 \ldots u_{k-1}v_0 \rightarrow u_2u_3 \ldots u_{k-1}v_0v_1 \rightarrow v_0v_1 \ldots v_{k-1} = v.$$ 

This walk might contain cycle(s), which suggests a shorter one is available. Therefore, the shortest distance between any two nodes is no greater than $k$. In terms of optical networks, the degree of a De Bruijn graph corresponds to the number of incoming/outgoing lightpaths and $k$ limits the number of hops required to reach any destination node.

### 2.2 Overview of Genetic Algorithms

Genetic algorithms are a class of randomized optimization heuristics loosely based on Darwin’s Evolutionary Theory. Since Holland [Hol75] and DeJong [Dej75] applied this idea into computer simulation, and supported its theoretical foundation with solid experimental data, GAs have been successfully used in enormously large areas including image processing and pattern recognition, job planning and scheduling, maze running and poker betting etc. [GE99, MNS99, Mor98, Ngo98, NY99, TMKH96]. GAs’ simplicity, flexibility, effectiveness and efficiency in search, optimization and machine learning are really quite amazing.

In GA, a candidate solution for a given problem is represented as an ordered list of values, called a chromosome and a group of chromosomes makes a population. Each position in a chromosome is called a gene and its value corresponds to an allele. A combination of several alleles forms a partial solution.

A genetic algorithm evaluates a population based on the application’s fitness function
and iteratively generates a new population. Each successive population is called a *generation*. Initially GA creates a generation G(0), and for each generation G(T), the algorithm creates a new one, G(T+1), through the generation procedure and evaluates it. Ideally after a number of generations, individuals in the population become alike to each other, reaching a state called *convergence*. Figures 2–2 and 2–3 outline the skeleton of GA and its generation procedure.

```
Generate initial population, G(0);
Evaluate G(0);
T: = 1;
Repeat
  Generate G(T) using G(T - 1);
  Evaluate G(T);
  T: = T + 1;
Until solution is found (convergence)
```

*Figure 2–2. An Abstract View of Genetic Algorithm*

*Figure 2–3. The Generation Procedure*
In these two graphs, *crossover* is a procedure where two chromosomes exchange their genetic materials to create offspring. *Mutation*, in computer simulation, is a procedure where a bit is randomly picked from a binary string and flipped from 1 to 0 or 0 to 1. *Reproduction* is a function where old chromosomes in a population are replaced by their offspring to create a new population. There are two types of reproduction plans: generational and state–steady, and the first one is used in the approach proposed in this thesis. Their definitions and other important concepts can be found in Appendix A.

### 2.2.1 Simple Genetic Algorithm and its Variations

A Simple Genetic Algorithm (SGA) is the simplest form of genetic algorithm with binary representation [Hol75], proportional selection [Dej75], generational reproduction [Sys91], one-point–crossover and random–bit–mutation [Hol75]. The example below illustrates how it works.

Assume there is a problem to find out a 6–bit string that contains mostly 1s.

Following steps in Figures 2–2 and 2–3, SGA proceeds as follows:

1. **Initialization**
   - Fitness Function: # of 1 in a 6–bit string;
   - Possibility of Crossover: \( P_c = 1.0 \);
   - Possibility of Mutation: \( P_m = 0.01 \);
   - Population Size: 4;
   - Create \( G(0) \) Randomly: 001101 110001 100010 111100

2. **Evaluate \( G(0) \)**
   - \( P_i \) is the probability of an individual being selected to mate.

   In proportional selection, \( P_i = \text{Fitness} / \text{Sum of Fitness} \).
<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Fitness</th>
<th>( P_r = \text{Fitness} / \text{Sum of Fitness} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>001101</td>
<td>3</td>
<td>3 / 12 = 0.25</td>
</tr>
<tr>
<td>110001</td>
<td>3</td>
<td>3 / 12 = 0.25</td>
</tr>
<tr>
<td>100010</td>
<td>2</td>
<td>2 / 12 = 0.17</td>
</tr>
<tr>
<td>111100</td>
<td>4</td>
<td>4 / 12 = 0.33</td>
</tr>
<tr>
<td>Sum of Fitness</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

3. **Generation Procedure for \( G(T) \)**

3.1 **Selection from \( G(T-1) \)**

Since \( P_c = 1.0 \), crossover always occurs. According to the \( P_r \) value, select two chromosomes from \( G(T-1) \) for mating. Assume 111100 and 110001 are selected.

3.2 **Crossover for the selected chromosomes**

Randomly pick a number from 1 to 5 as the crossover point. Assume 3 is picked:

\[
111 | 100 \rightarrow 111001 \\
110 | 001 \rightarrow 110100
\]

The two strings exchange genetic material from the cut point to create offspring. Therefore, 111 (from the first part of string1) is combined with 001 (from the second part of string2) to create offspring 111001. And 110 (from the first part of string2) is combined with 100 (from the second part of string1) to create 110100. Thus, two new chromosomes are created.

3.3 **Mutation for the new chromosomes**

Based on \( P_m = 0.01 \), assume 110110 does not mutate while 111001 does and 4 is picked randomly as the mutation position (range from 1 to 6):

\[
111001 \rightarrow 111101
\]

Therefore, after selection, crossover and mutation, two new offspring are created:

111101 and 110100.
3.4 Reproduction for G(T)

Since SGA uses a generational reproduction plan where the old generation is replaced by a brand new one, steps 3.1 to 3.4 have to be repeated until in total 4 offspring are created (remember the population size is 4).

4. Evaluation for G(T)

Similar to Step 2, evaluate the new population based on the given fitness function and calculate the selection probability \( (P_s) \) for each individual.

5. Check the stop criteria

If the stop criteria are not satisfied, repeat from Step 3 with \( T = T+1 \). In an ideal situation, when the program stops, chromosome 111111 should be found as the solution.

How to stop a GA program, so-called stop criterion (evaluation function for GA performance), is actually a very interesting question and sometimes is quite challenging too. In general, there are two types of stop criteria:

§ Best-one approach focuses on whether the best fitness is achieved and how fast it is achieved;

§ Majority rule is concerned about whether the population is converged, how stable the convergence is, and how close the average fitness is to the best fitness when converged.

However, the best-one approach (some call it off-line performance [Dej75, Mau84]) is impractical because for most cases people are working on, the best fitness might not be known. On the other hand, the majority rule (also called on-line performance [Dej75, Mau84]) is not perfect either because premature convergence [Dav90] might
occur. Even the combination of the two (when majority rule is met, the best fitness in
the population should be reasonably acceptable) is not ideal because for multiple-
peak problems, contention among the best chromosomes [FM93, MFH91, SEO91]
might trap the average fitness in an area far from the global optimum, which is called
stagnation. These kinds of challenges leave this area still open for further study.

There can be many variations to the SGA described above. These include having
different stop criteria, different population sizes, different reproduction plans,
different genetic operators and different chromosome representations.

§ Different Stop Criteria: the program can stop when the optimum is found (off-
line performance) or when the average fitness is close to the best fitness (on-line
performance);

§ Different Population Sizes: an unsettled issue that causes the exploration and
exploitation dilemma (Appendix A). Although some suggest (log N) (N is the size
of the search space) [Ala92], users still have to try different sizes before settling
down [Bry96, Gol89, Mic92];

§ Different Reproduction Plans: the whole population can be replaced for each
generation (generational) or just part of it (state–steady) [Sys91, TMKH96];

§ Different Genetic Operators: there are many types of crossovers and mutations
such as one-point-crossover, universal-crossover, flip-mutation, swap-mutation
etc. [Dej75, Esh91, Gol89, SD91];

§ Different Chromosome Representations: a chromosome can be represented as a
binary string, a gray-code string, a real number or a permutation etc. [Ant91,
Bou91, Gol89, Koz91, TMKH96];

§ Different Selection Schemes: such as proportional, tournament, ranking, elitist, or
universal selection [Dej75, Gol89, GD91, TMKH96, TG94].

In addition to variations in these GA parameters, it is also possible to combine selection, reproduction, crossover and mutation in different forms [Par93], such as selection plus crossover, or all of them except mutation, or mutation only, in which case GA works like Hill Climbing.

There is not much theoretical evidence showing which one of the above variations or combinations is the best. The evaluation of their performance is mainly based on empirical data. Therefore, for a specific problem, different strategies must be tried before making a final decision.

2.2.2 The Power of Genetic Algorithms

The working mechanism of GA is similar to Simulated Annealing (SA) [SH87] in the sense that they both use guided random search technique, but GA operates at a whole new level and provides unique flexibility, simplicity and robustness for many problems including those that are NP-hard. A discussion of conventional search techniques can be found in Appendix C. GAs differ from these conventional search schemes in four ways [Can95, Dej75, FM93, Gol89, Mic92, TMKH96]: data representation abstraction, parallel search, black box search and probabilistic rule.

2.2.2.1 Data Representation Abstraction

Unlike other methods that deal with functions and their control variables directly, genetic algorithms work with a coding of the parameter set, not the parameters themselves. GAs require the natural control parameters of the optimization problem to be coded as a finite-length string over a finite set of alphabets.
Consider a black box with 6 switches. For every setting of the six switches \( s \), there is an output signal \( f, f = f(s) \). The objective of the problem is to maximize the \( f \) value. The first thing a GA will do is to code \( s \) as a 6-bit string, where 1 means on and 0 means off. In such case, the job is to find a 6-bit string with maximum \( f \) value, without worrying about the transition rules and the working mechanism of the black box.

Consider another case to maximize \( f = (x^2) \) in the integer interval of \((0, 31)\), where \( x \) is coded as a 6-bit binary. Compared to the black box problem, this one has no difference in the abstract level except the objective function, which means the same GA mechanism can be used to solve both of them.

Therefore, because of the abstraction of data representation from the specific problem and the operation on the coding level, GAs can exploit coding similarities in a very general way: similar genetic operators, sometimes even the whole algorithm, can be applied in different problems, provided that they use the same coding method. As a result, GAs are largely unconstrained by the limitations that trouble other conventional methods like continuity, derivative existence, unimodality and so on.

Bit (binary) string is the most popular coding method in GAs because it is simple and heavily studied. But this might not be sufficient. [Wri91] points out that on some occasions real parameter coding outperforms binary-coded GA. And some other occasions like TSP [Mic92], classifier system [BGH87] and the prisoner dilemma [For85, Gol89], are naturally not suitable for bit string to handle. Other coding approaches include order-based representation, embedded lists and variable element
lists etc. [Mic92].

There are two basic principles of coding, as suggested by Goldberg [Gol89]:

§ Principle of meaningful building block: A coding should create short, low–order schemata relevant to the underlying problem and relatively unrelated to "the schemata over other fixed positions".

§ Principle of minimal alphabets: Users should choose the smallest alphabet set that permits a "natural expression" of the problem.

2.2.2.2 Parallel Search

Many optimization methods start from a single point in the target space and move to the next one determined by some transition rules. These point–to–point methods are not only insufficient but also dangerous because they are easily fooled by local optimums in multimodal search spaces. On the contrary, GAs work from a population, climbing many peaks simultaneously, which is well known as "implicit parallelism" [Dej75, Gol89, Hol75, TMKH96]. In general, the rich set of points in a population reduce the probability of locating the false peak to minimum.

As an example, consider the black box problem again. When GA starts, it randomly creates a group of strings (assume the population size is 6):

001011, 111001, 101001, 110001, 000101, 111101.

Afterwards, successive populations are generated using selection and reproduction until the stop criteria are satisfied, i.e. the best solution is found. Therefore, by working from a population with good diversity instead of a single point, GA finds "safety in numbers" [Gol89] and finds it fast.
2.2.2.3 Black Box Search

Many search techniques require auxiliary information in order to work properly. For example, gradient search is unable to climb the hill without the help of derivatives; greedy technique requires knowledge of "the tabular parameters" [SDK83] to complete its search.

By contrast, GA is blind. GA is a black-box algorithm as it optimizes a function using a strategy essentially independent of the task at hand. A genetic algorithm does not have to understand the problem domain to perform an effective search. In principle, GAs can work on any problems as long as they can be coded and can provide an objective function to calculate the payoff values associated with individual strings.

Intuitively, when GA works on a specific problem where such non-payoff information is available, it might be unwise not to use it. Therefore, some researchers presented knowledge-directed genetic algorithms [GGRG85, Gol89, Mic92] to exploit such opportunities. Nonetheless, GA's minimum requirement for extra knowledge of the tasks makes it a more generally used algorithm and broadens its application domain.

2.2.2.4 Probabilistic Rule

Different from deterministic search methods, GA uses probabilistic transition rules to guide its search. As discussed early in Section 2.2.1, mutation, selection, crossover and other advanced genetic operators are totally randomized: which one will occur, when will occur and where to occur are fully based on possibilities. Their
computation results are unpredictable to some degree. But the use of probability theory does not suggest this method is some simple random search. It is not decision making at the toss of a coin. "While randomized, genetic algorithms are no simple random walk; they efficiently exploit historical information to speculate on new search point with expected improved performance" [Gol89]. GAs inherit the theory of survival of the fittest and use random choice (randomized genetic operators) as a tool to lead a search toward regions of the search space with likely improvement.

Although GAs are powerful search tools, they do have certain limitations. A GA does not guarantee that it can find the global optimum, nor does it guarantee the proper convergence in arbitrary problems. In general, a customized technique would probably outperform a GA on a specific problem. The main advantage of GA is that for an arbitrary problem, it can find a reasonable solution within reasonable time, without auxiliary information and without being customarily designed for the problem.

![Diagram](image)

**Figure 2–4. Comparison: A General Perspective**

In conclusion, GAs are robust, efficient and applicable to varied domains. Figure 2–4
from [Gol89] illustrates the comparison of GAs' performances across different problem spectrums with other methods'. Note that a robust GA works very well across different problem domains; specialized technique works very well too, but only in a narrow area (unimodal problems); and both enumerative and random walk methods are less efficient than a GA.

### 2.2.3 Transformation

In many problems, the objective is naturally stated as the minimization of some cost function \( g(x) \) rather than the maximization of some profit function \( u(x) \). Even if the problem is about maximization, this alone does not guarantee that the profit function will be nonnegative for all \( x \). But a GA requires a fitness function which tries to maximize a nonnegative value (fitness). Therefore, some mechanism is needed to map an objective function to a fitness function. An objective function is problem-specific. It could be in minimization or maximization form and its value could be positive, zero or negative. By contrast, the fitness function required by a GA is always in maximization form and can only have nonnegative values.

To transform cost minimization to fitness maximization without negative, the following equation is commonly used [Gol89]:

\[
    f(x) = C_{\text{max}} - g(x) \quad \text{where } g(x) < C_{\text{max}}
\]

\[
    = 0, \quad \text{otherwise.}
\]

To offset the negative value in a maximization problem, the transformation equation is [Gol89]:

\[
    f(x) = C_{\text{max}} + u(x) \quad \text{where } u(x) + C_{\text{max}} > 0
\]

\[
    = 0, \quad \text{otherwise.}
\]
In these two equations, \( C_{\text{max}} \) may be taken as an input coefficient, as the largest \( g/u \) value observed so far, or as the largest \( g/u \) value in the current population, or as the largest of the last \( k \) generations. Or \( C_{\text{max}} \) can be calculated by using population variance [Gol89].

There are two other special types of transformations. One of them is called scaling which transforms a steep fitness function to a smooth one or vice versa. At the start of GA runs, when a fitness function is very steep, it means that several chromosomes have extraordinary fitness values in the initial population. These chromosomes would have significantly higher chances being selected for mating (remember the probability of selection is proportional to the fitness value of the individual). As a result, these chromosomes would dominate the whole population in several generations, i.e. premature convergence occurs. Therefore, some kind of transformation is needed to smooth this steep function.

On the other hand, if the fitness function is very smooth, it means that fitness values of chromosomes are distributed in a narrow area and the average fitness is close to the best fitness of the population (but not the global optimum). Selection has no pressure. The survival of the fittest for improvement becomes a random walk, a phenomenon called stagnation. In such case, the fitness function needs to be scaled up. Scaling technique includes linear scaling [Bag68], Sigma truncation [For85] and power law scaling [Gil85].

The other special transformation is called uniformity (or normalization) which maps fitness value to a uniform range in order to compare a GAs' performances over
various problem domains. Tang et al. introduce windowing and linear normalization to undertake this task [TMKH96]. All of the technical details about special transformations are left in the corresponding references.

2.2.4 Complement

Diversity is extremely important for a genetic algorithm. Poor diversity does not just slow down a GA’s parallel search but also easily traps it into a local optimum. Mutation is one way to introduce diversity into a population. However, high probability of mutation not only makes a GA shaky (the average fitness is up and down all the time) but also increases the possibility of accidentally destroying good chromosomes or good genes. And this is exactly where the complement operator [Mau84] steps in. By guaranteeing the presence of both good genes and bad genes in the population, the complement operator helps maintain a good diversity in the population and prevents premature convergence. And in principle, selection and crossover can get rid of those bad genes through the generation procedure.

If $S_r$ is a good chromosome (with high fitness value), intuitively $S_r$, its complement, would be the bad one. $S_r$ might accidentally be destroyed in one generation but its counterpart $S_r$ still exists. In the next generation, $S_r$ might be selected to mate. Its complement, which is $S_r$, would be recreated again. Therefore, complement operator reduces the possibility of destroying the good chromosomes. Moreover, the presence of complementary individuals makes a GA more resistant to deception since the best chromosome is naturally the complement of the deceptive optimal chromosome and vice versa. Experiments in [Mau84] proved this theory.
Complement is a symmetric operator: the complement of $S$ is $S^c$ and vice versa.

Complement for bit string is straightforward. An *exclusive OR* (XOR) with $11...1$ would serve. For example, the complement of $1110110$ will be

$$1110110 \text{ XOR } 1111111 = 0001001.$$  

But for other representations like permutation, the implementation of complement would be very different and difficult [Mic92].
CHAPTER 3

New Approach: A GA for the Mapping Problem

The main objective of this thesis is to study the use of genetic algorithms for designing the logical topology of multihop optical networks. The logical topology to be designed is represented by a directed graph $G_{d}(N, A_{d})$, where $N$ is the set of nodes and $A_{d}$ is the set of lightpaths (logical links) established over the physical network. The input to the algorithm is the traffic matrix $T = (t_{sd})$, where $t_{sd}$ represents the arrival rate of packets at node $s$ destined for node $d$. A link $(i, j) \in A_{l}$, if there is a logical link between nodes $i$ and $j$. Other important parameters are defined as follows:

§ $\lambda_{ij} = \text{arrival rate of packets over link } (i, j) \text{ from all source-destination pairs, i.e. } \lambda_{ij}$ is the load on link $(i, j)$.

§ $\lambda_{\text{max}} = \max \left\{ \lambda_{ij} \mid (i, j) \in A_{l} \right\}$ is defined as the congestion of the network.

This thesis has used the De Bruijn graph as the target logical topology because of its attractive properties such as low diameter, rich interconnection and a simple routing scheme. Since the target logical topology is already known, the topology design problem is reduced to find an appropriate mapping between the nodes of the physical network and the nodes in the logical topology. A routing scheme over the topology also needs to be defined, which tries to reduce the congestion of the network. This chapter introduces a customized genetic algorithm for the logical topology design problem and discusses various control parameters, the chromosome representation
scheme, the objective function, and the stop criteria.

The basic structure of the algorithm follows that of the SGA [Gol89] and is given in Figure 3–1. Here G(0) represents the initial population and G(T) is the population in the Tth generation. The rest of this chapter will discuss in detail the steps outlined below.

<table>
<thead>
<tr>
<th>Initialization;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set up all control parameters</td>
</tr>
<tr>
<td>Calculate the alternative paths</td>
</tr>
<tr>
<td>Create first population, G(0);</td>
</tr>
<tr>
<td>Evaluate G(0);</td>
</tr>
<tr>
<td>T: = 0;</td>
</tr>
<tr>
<td>Repeat</td>
</tr>
<tr>
<td>T: = T + 1;</td>
</tr>
<tr>
<td>while G(T) is not full</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>Select chromosomes from G(T − 1);</td>
</tr>
<tr>
<td>Apply crossover and/or mutation;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>Evaluate G(T);</td>
</tr>
<tr>
<td>Until Stop Criteria are satisfied</td>
</tr>
</tbody>
</table>

Figure 3–1. The Proposed Genetic Algorithm

3.1 Chromosome Representation

In the proposed algorithm, each potential solution (or chromosome) is a mapping from physical nodes to logical nodes. Each chromosome can be represented uniquely by an ordered list, where the positions in the list represent logical node numbers and
the corresponding values are the physical nodes, to which these logical nodes are mapped.

As an example, consider a chromosome for an 8-node De Bruijn graph \((d=2, k=3)\) shown below. The top row gives the logical node number (and associated label) and the second row shows the physical node to which each logical node is mapped.

<table>
<thead>
<tr>
<th>Logical node:</th>
<th>0 (000)</th>
<th>1 (001)</th>
<th>2 (010)</th>
<th>3 (011)</th>
<th>4 (100)</th>
<th>5 (101)</th>
<th>6 (110)</th>
<th>7 (111)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical node:</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>D</td>
<td>G</td>
<td>H</td>
<td>F</td>
</tr>
</tbody>
</table>

The actual mapping corresponding to this chromosome is shown in Figure 1-3 in Section 1.3. In order to avoid confusion, the remainder of this thesis uses numbers to represent logical nodes and alphabets to represent physical nodes. However, in the actual implementation, both are represented by integers. It is clear that if the logical nodes are always presented in the same order as shown above, a chromosome can be uniquely identified by an ordered list of the physical nodes. The logical node to which a physical node is mapped is simply determined by its position in the list. So, the list \((B \ A \ C \ E \ D \ G \ H \ F)\) represents the same chromosome as the one given above. This thesis uses both representations interchangeably.

3.2 Initialization

All control parameters are set up during the initialization phase, including the initial population and population size, the possibilities of mutation and crossover, the crossover strategies to be used etc. During experiments, the population size has been varied depending on the size of the network and the performance of the genetic algorithm. A number of different crossover strategies have also been tried. The details
of the crossover strategies are explained in Chapter 4. Once the control parameters have been initialized the algorithm randomly generates the appropriate number of chromosomes for the initial population G(0) and evaluates each chromosome using the objective function.

3.2.1 Alternative Path Algorithm

During the initialization phase, the proposed algorithm also calculates several node disjoint paths, over the logical topology, between every source–destination (s–d) pair. Communication between each s–d pair is routed over one of these alternative paths. There are a number of alternative paths (instead of only the shortest path) considered in the algorithm because it allows more flexibility in routing each communication. If routing over one of the (longer) alternative paths results in a lower value of congestion, that path is selected. Thus, the algorithm is willing to tolerate a limited amount of additional delay (corresponding to longer alternative paths) in order to reduce congestion. One of the reasons for selecting the De Bruijn graph as the logical topology is its rich connectivity. It has been shown that for a De Bruijn graph of degree d and diameter k, there are at least \( d-k \) node disjoint alternative paths, with maximum length \( k+1 \) between each s–d pair [SR91, SR94]. Since all these paths are relatively short (O(log N)), by considering only these alternative paths, extra delay may not be incurred to a great extent. Also, since the paths are node disjoint, if one path traverses a high–traffic link, the others are guaranteed to avoid that link.

This section outlines an algorithm to find the alternative paths between two nodes \( N_1 \) and \( N_2 \) in the De Bruijn topology. Note that by considering even longer paths, it may be possible to reduce the congestion further. But such additional paths are not
considered for two reasons:

i) In order to maintain an upper bound on the path length.

ii) In order to reduce the complexity of the algorithm.

The objective, here, is to find a reasonable solution within reasonable time, rather than trying to find an optimal solution, which is also the major feature of GA.

1. Find all successors of $u$:
   
   $$S(u) = \{ \mathit{w} \mid \mathit{w} = u_1 \ldots u_{k-1} \mathit{i}, i \in (0 \ldots d-1) \}.$$  

2. Calculate the shortest path:
   
   § Calculate the longest matching prefix of $\mathit{v}$ and suffix of $u$,
   
   $$u_{k+t-1} \ldots u_{k-1} = v_0 \ldots v_t, t \in (0 \ldots k-2);$$

   § Repeatedly left-shift $v_{t-1} \ldots v_{k-1}$ to $u$ to create the shortest alternative path;

   § $S(u) = S(u) - \{ u_1 \ldots u_{k-1} \mathit{v}_{t+1} \}.$

3. Calculate the other paths:

   For all $\mathit{w} \in S(u)$

   § Calculate the path from $\mathit{w}$ to $\mathit{v}$ by left-shifting $v_0 \ldots v_{k-1}$ to $\mathit{w}$;

   § Add $u$ to the beginning of the path to construct the complete alternative path from $u$ to $\mathit{v}$.

   § $S(u) = S(u) - \{ \mathit{w} \}.$

4. For each path $P$ calculated in Step 3, check for cycles in $P$:

   § if cycle at the beginning of path $P$
     
     discard $P$;

   § if cycle in the middle of path $P$
     
     remove cycle to obtain a shorter path $P'$.

Figure 3–2. Alternative Path Algorithm

Let $G = B(d, k)$ be a De Bruijn graph of degree $d$ and diameter $k$. Let $u = u_0 \ldots u_{k-1}$ and $v = v_0 \ldots v_{k-1}$, be two nodes in the network such that $u \neq v$. In order to find the
alternative paths from $u$ to $v$, the algorithm first finds the shortest one, which is unique and is of length $k-t$ where $t$ is the length of the longest suffix of $u$ that appears as the prefix of $v$. Then it calculates the remaining paths of length $k+l$. If there are cycles in any of these paths, the cycles must be eliminated. The algorithm is outlined in Figure 3–2 for finding up to $d$ alternative paths between two nodes $u$ and $v$ in the logical topology.

For example, consider paths from node 1 (01) to node 3 (12) in a $3^2$ De Bruijn graph. Using the above algorithm, three alternative paths can be created. The shortest one is

$$01 \rightarrow 12.$$ 

The two paths calculated in Step 3 are:

$$P_1 = 01 \rightarrow 10 \rightarrow 01 \rightarrow 12 \quad \text{and} \quad P_2 = 01 \rightarrow 11 \rightarrow 11 \rightarrow 12.$$ 

$P_1$ is discarded because the cycle $01 \rightarrow 10 \rightarrow 01$ occurs at the beginning of the path.

$P_2$ is modified, by removing the cycle $11 \rightarrow 11$, which occurs inside the path. This results in a new and shorter path $P_2' = 01 \rightarrow 11 \rightarrow 12$.

3.3 Selection, Crossover and Mutation

The selection strategy of the proposed GA is similar to the one used in SGA, i.e. proportional selection, where the probability of being selected for a chromosome is equal to the fitness of the chromosome divided by the sum of the fitness in the population. **Elitist** selection is also incorporated into the approach. This means that in every generation, a certain number of chromosomes with the highest fitness value automatically enter the next generation $G(T)$ without going through the selection and mating procedure. This approach is also known as pre–selection [Gol89]. With the elitist approach, the best chromosome always enters the next generation unchanged.
This means that the best fitness value can never decrease from one generation to the next. One important factor that can affect the performance of the proposed genetic algorithm is the percentage of chromosomes that are pre-selected. If it is too low, good chromosomes in the previous generation can not be protected. If too high, early convergence might occur; or chromosome contention could become very strong (Chapter 5) resulting in stagnation. As suggested by [Dej75, Gol89], 1% is chosen as the pre-selection value, which means chromosomes on the top 1% of the population advance to the next generation without any changes.

The mutation operator in the proposed algorithm is a simple swap operator, where two genes in a chromosome are randomly picked and switch their values. For example, the following is a chromosome corresponding to an 8-node network.

<table>
<thead>
<tr>
<th>Logical node:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical node:</td>
<td>B</td>
<td>A</td>
<td>D</td>
<td>E</td>
<td>C</td>
<td>G</td>
<td>H</td>
<td>F</td>
</tr>
</tbody>
</table>

If genes in position 3 and 6 are picked, after mutation the new chromosome will be:

<table>
<thead>
<tr>
<th>Logical node:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical node:</td>
<td>B</td>
<td>A</td>
<td>D</td>
<td>H</td>
<td>C</td>
<td>G</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

The probability of mutation in the proposed GA is typically 1%. Under certain special circumstances, if the algorithm suspects premature convergence, it increases the mutation rate to 51% (for one generation only) in order to introduce diversity into the population. The probability of crossover is 100% (suggested by [Gol89]), which means crossover always happens. This research has used a number of different crossover strategies and they are detailed in the next chapter.
3.4 Objective Function and Fitness Value

In GA, an objective function is used to calculate the objective value of a chromosome. In this research, the chromosome represents a mapping between the physical and logical nodes and the objective value is the congestion for that mapping. The value of congestion is dependent on how the communications are routed over the topology. Here a greedy heuristic is used, where communications with the highest traffic are considered first. For each communication, only up to $d$ alternative paths (as discussed in Section 3.2.1) are considered and the communication is routed over the path with the least traffic. Since this is a greedy heuristic, once a communication has been routed over a particular path, it is final. Subsequent routing decisions for other communications will depend on the previous decisions.

1. Initialize
   \[
   \lambda_{ij} = 0 \text{ for all } (i, j) \text{ in } A_i
   \]
   Congestion = 0
   Threshold = $T_m$
2. Pick highest traffic entry $t_{st}$ from traffic matrix $T$
   If $t_{st} < T_m$, go to Step 8
3. $P =$ set of Alternative paths between $s$ and $d$
4. Select $P_k$ in $P$, where $P_k$ is the path with the least heavy traffic
   If more than one path with same traffic, pick shortest path
   If traffic and path length be the same, pick one randomly
5. For each link $(i, j)$ in $P_k$
   \[
   \lambda_{ij} = \lambda_{ij} + t_{st}
   \]
6. In traffic matrix $T$, set $t_{st} = -t_{st}$
7. Repeat steps 2–6
8. Congestion = $\max\{\lambda_{ij} \mid (i, j) \text{ in } A_i\}$

Figure 3–3. Outline of Objective Function
In order to reduce the complexity, this research has not considered the traffic between all \( s-d \) pairs when calculating the congestion. It has only considered communications for which the traffic is higher than a predefined threshold value. Figure 3–3 gives a brief outline of the algorithm for calculating the objective value for a single chromosome. The threshold is usually set to be 10%–30% of the maximal traffic.

The algorithm is illustrated through an example, which calculates the objective value (congestion) of chromosome \( C_1 \) (shown below), corresponding to an 8-node De Brujin graph with a traffic matrix \( T \).

Example 3–1

The threshold \( T_\alpha = 5 \); The traffic matrix:

\[
\begin{array}{cccccccc}
& A & B & C & D & E & F & G & H \\
A & 0 & 1 & 3 & 7 & 2 & 2 & 6 & 2 \\
B & 4 & 0 & 4 & 2 & 4 & 1 & 5 & 5 \\
C & 2 & 2 & 0 & 4 & 2 & 3 & 2 & 2 \\
D & 1 & 3 & 6 & 0 & 3 & 3 & 2 & 2 \\
E & 1 & 1 & 2 & 2 & 2 & 3 & 0 & 0 \\
F & 3 & 5 & 5 & 1 & 4 & 4 & 6 & 6 \\
G & 2 & 1 & 4 & 2 & 2 & 5 & 0 & 0 \\
H & 2 & 1 & 4 & 2 & 1 & 5 & 0 & 0 \\
\end{array}
\]

Table 3–1. Traffic Matrix for Example 3–1

Chromosome \( C_1 \):

<table>
<thead>
<tr>
<th>Logical node:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical node:</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>D</td>
<td>G</td>
<td>H</td>
<td>F</td>
</tr>
</tbody>
</table>
De Bruijn graph:

The algorithm for calculating the objective value works as follows:

1. Initialize
   
   Congestion = 0, \( T_n = 5 \);

2. Pick the highest traffic element \( t_{ab} = 7 \), from \( T \). From \( C_i \): \( A = 001, D = 100 \)
   
   \( t_{ab} > 5 \);

3. \( P = \{ P_1, P_2 \} \), where \( P_1 = 001 \rightarrow 010 \rightarrow 100; P_2 = 001 \rightarrow 011 \rightarrow 110 \rightarrow 100; \)

4. Since traffic on both \( P_1 \) and \( P_2 \) are same \( (= 0) \), pick shortest path \( P_1 \);

5. Update traffic on each link in path \( P_1 \) (Figure 3–5);
6. In the traffic matrix $T$, change $t_{ab}$ from 7 to −7 indicating that it has been processed;

2. Repeat from Step 2, pick $t_{ag} = 6$, from T. From C; A = 001, G = 101

$t_{ag} > 5$;

3. $P = \{P_1, P_2\}$, where $P_1 = 001 \rightarrow 010 \rightarrow 101$; $P_2 = 001 \rightarrow 011 \rightarrow 110 \rightarrow 101$;

4. Traffic on link 001 → 010 of $P_1$ is 7; traffic on all links of $P_2$ is 0, pick $P_2$;

5. Update traffic on each link in path $P_2$ (Figure 3–6);

6. In the traffic matrix, change $t_{ag}$ from 6 to −6 indicating that it has been processed;
In a similar way, process all entries in the traffic matrix which are greater than the threshold value. And the objective value (congestion) is equal to the maximal traffic offered to a link in the De Bruijn graph.

3.4.1 Fitness Value

The objective of the proposed GA is to design a topology and the associated routing scheme, which minimizes the congestion. However, a GA typically attempts to maximize its fitness value. Therefore, a function is needed to transform congestion (objective value) to a non-negative fitness value. The function given below transforms the objective value for a chromosome \(x\) (objective\(x\)) into the corresponding fitness value (fitness\(x\)).

\[
\text{fitness}(x) = 0, \text{ where MaxCongestion < objective}(x) \\
= \text{MaxCongestion} - \text{objective}(x), \text{ otherwise}
\]
MaxCongestion is the largest congestion observed so far during the execution. Initially MaxCongestion equals ZERO. For each generation, after calculating the objective values (congestion) of the whole population, the algorithm checks whether the highest congestion of this generation is greater than MaxCongestion. If so, it updates MaxCongestion, before calling the above function to convert objective values to fitness values.

One drawback of this transformation function is that the fitness value is dependent on the results from the previous generations. This could cause early convergence or trap the algorithm into a local optimum. One solution is to make the population size sufficiently large. But a better idea is using population variance [Gol89].

3.5 Stop Criteria

A stop criterion is the condition under which a GA should stop. Many different stop criteria have been suggested in the literature, such as on-line performance, off-line performance, hamming distance, fixed number of generations and fitness deviation. The algorithm implemented in this thesis is a combination of on-line performance and fixed number of generations. The definition of convergence is that chromosomes in a population become identical or similar to each other. Typically, a GA can be considered have converged if the average fitness of the population (AvgFitness) is at least 90% of the best fitness (BestFitness) – it is called the 90%-rule.

However, for multimodal problems, even if a population’s average fitness is close to its best fitness, it does not necessarily mean that its chromosomes become similar to
each other. To illustrate such situation, consider an example of maximizing the
function \( f(x) = x^2 \), where \( x \) is an integer from \(-10\) to \(10\) inclusive. Obviously there are
two different chromosomes, \(-10\) and \(10\), can both reach the global optimum \(100\). If
somehow half of the population are 10s and the other half are \(-10s\), its average fitness
and best fitness will both be equal to \(100\). According to the 90\% rule, the algorithm
is considered converged. But all the chromosomes have not become identical or
similar.

Although the mapping problem presented in this thesis is a multimodal problem,
according to the experimental data, whenever the average fitness is at least 90\% of the
best fitness in the population, the chromosomes always become identical or similar to
each other. So, in this case, the 90\% rule is still applicable and it has been used to
determine convergence.

```
if (G ≥ MaxGen)
    Program stops and algorithm fails;
if (MinGen < G ≤ MaxGen) && (AvgFitness ≥ 0.9 * MaxFitness)
    Program stops and algorithm succeeds;
if (G ≤ MinGen) && (AvgFitness ≥ 0.9 * MaxFitness) //Early convergence
    Pmutate += 0.5 //increase probability of mutation
```

Figure 3-7. Stop Criteria

In order to minimize the possibility of premature convergence, the program has to go
through at least a minimal number of generations (MinGen) before it stops. If the
algorithm finds that the 90\% rule is applicable prior to that, it increases the
probability of mutation (Pmutate), in one generation, to introduce more diversity into
the population. Finally, if the program executes a pre-specified maximal number of iterations (MaxGen) and still does not converge, the program stops and it is considered a failure. Since there is not much theoretical foundation for setting the minimal and maximal number of generations, they have been set based on observation and empirical data. The outline of the stop criteria is given in Figure 3-7. Here, $G$ is the number of generations gone through.

One thing must be pointed out that convergence actually does not mean anything in the presented mapping problem as long as a good solution can be found. However, the principle of GA, survival of the fittest, dictates that GAs should always converge. Therefore, the research includes it as one of the stop signals.

3.6 Conclusion

This chapter proposes a tailor-designed genetic algorithm to solve the multihop optical network design problem. It outlines its structure, presents the objective function, and explains the working mechanism. The next chapter introduces different crossover strategies used. But the questions still remain unanswered: can this GA work and how well does it perform? In Chapter 5, the correctness of this algorithm as well as its efficiency are going to be tested with a series of experiments.
CHAPTER 4

Crossover Strategies

The crossover operator exchanges genetic material between two chromosomes and assembles short, low order and highly fit building blocks together to produce better and better solutions. This chapter first reviews Order Crossover, which has been proposed for the TSP problem, and then introduces several new crossover strategies designed specifically for the logical topology design problem. Their working mechanisms as well as their strengths and weaknesses are explained in detail.

4.1 Order Crossover

The Order Crossover strategy (OCX) was introduced by Davis [Dav85] for the Traveling Salesman Problem (TSP). It is borrowed in this thesis as the starting point. The original idea of OCX is to create an offspring by choosing a sub-sequence of a tour from one parent and preserving the relative order of cities from the other parent.

As an example, let $C_1$ (Table 4–1) and $C_2$ (Table 4–2) be the chromosomes chosen for crossover. Since OCX does not care about index and label, $C_1$ and $C_2$ can be simply written as (B A C E D G H F) and (C D A E F G H B).

\[
\begin{array}{cccccccc}
\text{Logical Node:} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\text{Label:} & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\text{Physical Node:} & B & A & C & E & D & G & H & F \\
\end{array}
\]

Table 4–1. Chromosome $C_1$
C1:

<table>
<thead>
<tr>
<th>Logical Node:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label:</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>Physical Node:</td>
<td>C</td>
<td>D</td>
<td>A</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>B</td>
</tr>
</tbody>
</table>

Table 4-2. Chromosome C1

Because the chromosome length is 8, two numbers are randomly picked from 1 to 7 as the two crossover points. Assume 2 and 6 are picked and their positions are marked by "|".

\[C_1 = (B \ A \ | \ C \ E \ D \ G \ | \ H \ F)\]

\[C_2 = (C \ D \ | \ A \ E \ F \ G \ | \ H \ B)\]

The offspring are produced in the following way. First the segments between the two cut points are copied into the offspring:

\[\text{Offspring1} = (? \ ? \ | \ C \ E \ D \ G \ | \ ? \ ?)\]

\[\text{Offspring2} = (? \ ? \ | \ A \ E \ F \ G \ | \ ? \ ?)\]

Next, starting from the second cut point of one parent, the nodes from the other parent are copied in the same order, ignoring symbols already present. When reaching the end of the string, continue from the head of the string. The sequence of the nodes in C2 (from the second cut point) is

\[H \rightarrow B \rightarrow C \rightarrow D \rightarrow A \rightarrow E \rightarrow F \rightarrow G\]

To fill in the blank marked by "?" in C1, nodes C, E, D, G, which are already there, need to be removed. As a result, the remaining sequence is

\[H \rightarrow B \rightarrow A \rightarrow F\]
This sequence is placed in Offspring1 starting from the second cut point:

\[ \text{Offspring1} = (A \ F \ C \ E \ D \ G \ H \ B) \]

Similarly the second offspring can be created:

\[ \text{Offspring2} = (C \ D \ A \ E \ F \ G \ H \ B) \]

### 4.2 Cluster Crossover

In the TSP problem, the building blocks are sub-tours, and OCX is a natural crossover strategy for TSP because it intends to construct and preserve optimal sub-tours. However, sub-tours have nothing to with the presented mapping problem. Therefore, crossover strategies that are more suitable for this application are need.

![Figure 4-1. An Example of Cluster Centered at 011](image)

In order to minimize congestion, two nodes should be placed as close to each other as possible if they have heavy traffic. It is recognized that the most important structure that needs to be preserved, is a node, surrounded by its adjacent members. This is identified as the basic building block and is called a cluster. In other words, a cluster
is a special sub-graph (Section 4.3) containing a given node and all its successors and predecessors. Figure 4–1 shows an example of a cluster, centered at node 3 (011), in an 8–node De Bruijn graph. The Cluster Crossover strategy (CCX) tries to preserve good clusters in the chromosomes.

The following steps describe, through an example, how CCX works. Chromosomes $C_1$ and $C_2$ are used again, given by Tables 4–1 and 4–2, to illustrate the procedure:

1. Randomly pick a node (011) as the center.

2. Protect all nodes belonging to the cluster centered at node 011. Therefore, Offspring1 would be:

<table>
<thead>
<tr>
<th>Logical Node:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label:</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
</tr>
</tbody>
</table>

3. Fill the remaining positions (marked by "?") with the genes from $C_1 = (C, D, A, E, F, G, H, B)$. After getting rid of those nodes already in Offspring1, (C, D, B) are left. Use them, from left to right, to replace the question marks in Offspring1. Thus, Offspring1 would be:

<table>
<thead>
<tr>
<th>Logical Node:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label:</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>Physical Node:</td>
<td>C</td>
<td>A</td>
<td>D</td>
<td>E</td>
<td>B</td>
<td>G</td>
<td>H</td>
<td>F</td>
</tr>
</tbody>
</table>

Applying the same strategy to $C_2$, Offspring2 can be created.

4.3 Sub-graph Crossover

In a De Bruijn graph of degree $d$, each node has up to $d$ predecessors and $d$ successors, i.e. up to $2d$ nodes adjacent to it. CCX protects all the nodes in a cluster. The Sub-graph Crossover strategy (SCX) is a modification of CCX in which a given
node \( n \) and \( p \) of its adjacent nodes are preserved, where \( 1 \leq p \leq 2d \). The number of adjacent nodes to be preserved is chosen by randomly picking a number from \( 1 \) to \( 2d \).

Since some nodes have only \( d-1 \) predecessors and \( d-1 \) successors, it is possible that a number, that is greater than the total number of nodes adjacent to \( n \), is picked. In such case, SCX simply protects the entire cluster and becomes identical to CCX. As in CCX, the unprotected positions in the first chromosome of SCX are filled with genes from the second chromosome.

As an example, let \( C_1 \) (Table 4-1) and \( C_2 \) (Table 4-2) be the chromosomes chosen for crossover. SCX creates the new offspring as follows:

1. Randomly pick a node as the center of the sub-graph. Assume node \( (011) \) is chosen.

2. Randomly pick a number \( p \) from \( 1 \) to \( 2d \), (i.e. from \( 1 \) to \( 4 \), since \( d = 2 \) in this case). Assume \( p = 3 \) is pick.

3. Node \( (011) \) has two predecessors, 101 and 001, and two successors, 110 and 111. SCX first protects the predecessors, then the successors. Therefore the center node \( (011) \), and its three adjacent nodes including two predecessors (101 and 001) and one successor (randomly pick 111) go to Offspring1:

<table>
<thead>
<tr>
<th>Logical Node</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label:</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
</tr>
</tbody>
</table>

4. Fill the vacant positions, marked by "?", with genes from \( C_2 \) (as in CCX). Thus, an offspring is created:

<table>
<thead>
<tr>
<th>Logical Node</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label:</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>Physical Node</td>
<td>C</td>
<td>A</td>
<td>D</td>
<td>E</td>
<td>H</td>
<td>G</td>
<td>B</td>
<td>F</td>
</tr>
</tbody>
</table>
5. For C₂, repeat Step 1 to 4 to create another offspring.

Although CCX is a special case of SCX, experiments in chapter 5 reveal that CCX actually provides slightly better results. One reason for this may be that building blocks based on sub-graphs are not stable. Rather than combining the "good" sub-graphs together to form a good mapping, SCX might accidentally destroy them instead because it only protects part of the whole adjacent structure. Such partial protection is biased because it treats adjacent members differently: some are unchanged while others are replaced. But in fact, all successors and predecessors of a node are equally important. Also, in general, there are more nodes being protected in CCX than in SCX. The percentage of nodes unchanged during a crossover is called the protection ratio. As shown later in Chapter 5, when the protection ratio becomes too low, the performance, particularly the convergence factor, will degrade significantly.

4.4 Random Crossover

CCX shows strong cohesion for small networks because it can protect many nodes during a crossover. However, overly strong cohesion might cause early convergence. Moreover, CCX always protects clusters. But the problem is that it is very difficult to predict whether the cluster being protected is a good one. What happens if a cluster has some bad genes? Under CCX, a cluster can only be replaced but not reconstructed. In other words, if a cluster causes a mapping to have high congestion because of one or two bad elements inside it, CCX can only replace it with other clusters, but not fix it. Therefore, in order to create a complete and good mapping, CCX has to wait for all of the good clusters to show up and then assembles them together. Such waiting period could be very long, particularly when the problem size
is large.

The next crossover strategy, Random Crossover (RCX) does not care about any special structure of the chromosome representation nor does it care about the problem itself. The idea of RCX comes from uniform crossover, also known as multiple-point crossover [Gol89], in SGA. In order to create offspring, RCX randomly picks some positions to exchange genetic material while the rest of the chromosome remains unchanged. Therefore, RCX can exchange any elements whether it is part of a cluster or not.

To demonstrate how it works, chromosomes \( C_1 \) and \( C_2 \) are used again.

\[
C_1 = (B \ A \ C \ E \ D \ G \ H \ F) \\
C_2 = (C \ D \ A \ E \ F \ G \ H \ B)
\]

Since the chromosome contains 8 elements, an 8-bit binary string is created. Each bit corresponds to one element. The value of the string is randomly generated. Assume a coin is flipped eight times, and head is assigned as 1 and tail as 0. As a result, a binary 01101001 is created. Using this string, a mask operation is applied to \( C_1 \) (Figure 4-2). If a bit is "1", the corresponding element stays unchanged in the offspring; otherwise it is gone.

\[
\begin{array}{cccccccc}
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
B & A & C & E & D & G & H & F \\
\downarrow \\
\end{array}
\]

Figure 4-2. The Mask Operation
Then the vacant positions marked by "?" are filled with genes from C2. The first offspring, Offspring1, is shown below:

\[ \text{Offspring1} = (E \ A \ C \ G \ D \ H \ B \ F). \]

Applying the same strategy to C2, the second offspring can be created:

\[ \text{Offspring2} = (C \ D \ A \ E \ F \ G \ H \ B). \]

On average, 50% of the nodes are protected under RCX.

4.5 Mixed Crossover

All above crossover operators have their advantages and disadvantages. Obviously it is impossible to execute them all at the same time. But if for each time entering the crossover routine, one of the above CX strategies is randomly picked and carried on one at a time. This might be able to combine their strength together and overcome their weaknesses. This is the main idea behind the Mixed Crossover strategy (MCX).

MCX assigns a constants (OCX = 0, SCX = 1, CCX = 2, RCX = 3) to each crossover strategy. Every time a crossover occurs, a number is randomly picked from 0 to 3. Based on this value, the corresponding crossover operator is then called and executed.

4.6 Complement Operator

Complement is not a stand-alone operator. It is an accessory to other crossover operators. As discussed in Section 2.2.4, the function of complement is to maintain diversity in a population and avoid early convergence. For a binary string, finding its complement is easy. But for permutation, which is the case in this thesis, and other
chromosome representations, it is very difficult. In this section, a new scheme is proposed to implement the complement operator for permutation chromosomes.

The working mechanism of the complement operator presented here is similar to that of RCX. If "0" is used to represent a gene changed and "1" means unchanged, all crossovers can be described as applying a mask operation to a binary string. The binary string is called the *binary mask* or *mask*.

As a demonstration, the example in CCX is used again. First pick Node, (011) as the center and find its four adjacent members: 001, 101, 110 and 111. Since the positions for these four elements are 1, 3, 5, 6, 7, a binary mask *m* is created with "1" filled in these positions, that is, *m* = 01010111. To simulate CCX, the binary mask is applied to *C*_1 just like what RCX does:

```
0 1 0 1 0 1 1 1
B A C E D G H F
↓
```

Then replace "?" with genes from *C*_2. Thus, Offspring1 is

(C A D E B G H F).

This result is the same as the one obtained by using CCX.

After this mask procedure, also called the *simulated crossover*, the next step is the complement operation. First, the complement of *m* is calculated, which is 10101000. This complement string is applied to mask *C*_1 just like what is done during the
simulated crossover except that the binary string is different:

```
1 0 1 0 1 0 0 0
B A C E D G H F
```

Then replace "?" with genes from C2. Thus, Offspring2 is

```
(B A C E D F G H).
```

Generally one crossover produces only one offspring for each parent. But a crossover accompanied by a complement operator can create two offspring corresponding to one parent chromosome.

### 4.7 Conclusions

This chapter introduces Order Crossover, a very successful genetic operator in the TSP problem. It also proposes several new mating operators, including CCX and SCX, which take advantage of the special structure of the target problem. On the contrary, RCX does not make use of the auxiliary information but it guarantees a 50% protection ratio regardless of the network size (more discussion in Section 5.1). MCX attempts to combine all the strength of the above crossover strategies. With the help of the complement operator, the algorithm can maintain good diversity in the population. The effectiveness of these new operators is tested in the next chapter.
CHAPTER 5
Experimental Results and Performance Analysis

This chapter attempts to investigate how useful the GA framework is for logical topology design. In order to study the effectiveness of the proposed genetic algorithms, a series of experiments is conducted. The first set of experiments studies the performance of alternative strategies such as Hill Climbing (HC) [Bel57, JW94] and Evolutionary Strategy (ES) [Bel57, Bel61], which are discussed in Appendix C. For most practical sized networks, it is very difficult to determine the globally optimal logical topology. Therefore, the artificial traffic matrix is introduced, for which the minimal congestion can be easily calculated. This thesis uses such minimal congestion and the results of HC and ES on random traffic matrix as benchmarks to evaluate the performance of the proposed GA. After presenting each set of experimental results, this chapter also includes a critical analysis of the results and discusses ways to improve the performance.

5.1 Test Environment and Control Parameters

C/C++ is used to implement the proposed algorithm and the program is tested in Windows98, Pentium 233 MMX with 64M memory. Unless specified, all test results presented here are the averages of 50 runs.

Experiments in this thesis focus on a number of key parameters and how they affect
the performance of the GA. These parameters include the crossover strategy used, the population size, the number of generations, and the type of traffic matrix. Two types of traffic matrices have been experimented. The first is a random traffic matrix, which, as the name suggests, randomly assigns traffic values between each source-destination pair. For most networks it is extremely difficult to determine the optimal logical topology. Therefore, this thesis evaluates the performance of the GA in terms of the following two factors, which are used during the OCX experiment:

i) How often the GA converges and

ii) The fitness value of the best chromosome with respect to those obtained by using HC and ES.

Ideally, it would be desirable to measure the performance of the GA based on the global optimum. However, considering that there are $N!$ mappings for a given traffic matrix and each one has at least $N^2$ flow assignments, it is very difficult to find the minimal congestion for a random traffic matrix. Therefore, the artificial traffic matrix is introduced, for which the minimal congestion can be easily determined.

An artificial matrix is constructed in such a way that, when the mapping (between logical and physical nodes) is done optimally, each node communicates only with its immediate neighbors (adjacent members) in the logical topology. Table 5–1 is an example of an artificial traffic matrix for a $3^2$ De Bruijn graph. The best mappings include $(A B C D E F G H I)$, $(I H G F E D C B A)$, and $(A C B G I H D F E)$, all with a minimal congestion of 60. In general, for a $d^e$ De Bruijn graph there are at least $d$ best mappings for each such artificial traffic matrix. Therefore, using the artificial traffic matrix does not change the nature of the problem — it is still a multiple peak problem.
The congestion values for different mappings based on an artificial matrix are distinguishably discrete. In this case, the values range from the minimal 60, to the second minimal 120, to the third minimal 180, then 240, 300 etc. An example of an optimal mapping for the traffic matrix of Table 5–1 is shown in Figure 5–1.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>–</td>
<td>60</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>–</td>
<td>0</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>–</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>D</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>–</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>–</td>
<td>60</td>
<td>0</td>
<td>60</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>–</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>G</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>0</td>
<td>–</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 5–1. An Artificial Traffic Matrix for 3^1 De Brujin Graph

![Figure 5–1. An Optimal Mapping for Traffic Matrix in Table 5–1](image-url)
When using the artificial traffic matrix, the performance of the GA is evaluated in terms of the following factors:

i) How often the GA converges;

ii) How often the GA finds the global optimum and

iii) How often it is able to find a good solution, i.e. the second or third best mapping, even if it can not find the best mapping.

The above standards are typically applied for experiments on SCX, CCX and RCX. Before moving on, several important concepts need to be introduced. The first one is the Coverage Ratio (CR), which is the percentage of the search space that an algorithm travels (evaluates) before getting a result. It is used to measure the efficiency of a search algorithm. An efficient algorithm has the ability to search a small space to get a good solution. With the same result, the lower the CR ratio, the better the algorithm is. In sequential genetic algorithm, the equation to calculate CR is given as

\[
CR = \text{traveled space} + \text{search space}
\]

The traveled space is described as the number of generations gone through multiplied by population size. And the search space is approximated as \(N!\) in the mapping problem, where \(N\) is the number of nodes in the network. Increasing the population size can increase the CR ratio and the quality of the search result is improved correspondingly. In an extreme case, when the traveled space equals the search space, the global optimum can always be reached, although it is inefficient and impractical.

Protection Ratio (PR) denotes the percentage of nodes that are unchanged during a crossover.

\[
PR = \frac{\text{No. of nodes unchanged during one CX}}{\text{No. of nodes in the network}}
\]
When PR is high, it shows strong cohesion and there is a large possibility of early convergence. When PR is too low, crossover shows severe fluctuation, resulting in stagnation. The average fitness of the population in each generation tends to follow the variations in the protection ratio.

The remainder of this chapter, unless otherwise stated, uses probability of crossover = 1.0 and probability of mutation = 0.01.

5.2 Results for HC, ES and Order Crossover

In order to set up a benchmark for comparison, Hill Climbing and Evolutionary Strategy are also implemented for the mapping problem and integrated with the proposed algorithm. Table 5–2 shows the results of using HC and ES for logical topology design. Note that MaxGen Unchanged represents the maximal consecutive number of generations that the best fitness is not changed. It is the typical stop criterion for HC and ES. In all the tests, HC and ES result in only a small improvement, about 10%, over the initial population. It appears that simple mutation, which is the main reproduction strategy for ES and HC, does not have the speed to travel a huge search space (N!) to find a good solution within a reasonable amount of time.

<table>
<thead>
<tr>
<th># of Nodes</th>
<th>PopSize</th>
<th>MaxGen Unchanged</th>
<th>PR</th>
<th>Improvement over first Generation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>10</td>
<td>100</td>
<td>75%</td>
<td>10%</td>
<td>~3s</td>
</tr>
<tr>
<td>16</td>
<td>30</td>
<td>100</td>
<td>88%</td>
<td>10%</td>
<td>~5s</td>
</tr>
<tr>
<td>27</td>
<td>30</td>
<td>100</td>
<td>93%</td>
<td>10%</td>
<td>~3m</td>
</tr>
<tr>
<td>81</td>
<td>30</td>
<td>100</td>
<td>98%</td>
<td>10%</td>
<td>~6m</td>
</tr>
</tbody>
</table>

Table 5–2. Results Using ES/HC
Table 5-3 shows the results of using the Order Crossover strategy. The minimal and maximal numbers of generations for all these tests are 200 and 400. The traffic matrix is the same as that used for HC and ES: a random traffic matrix. The improvement and the running time are approximations based on observation, which is also applicable for the rest of the literature. Since the protection ratio for OCX is always 50% regardless of the network size, it is not shown in the table.

<table>
<thead>
<tr>
<th># of Nodes</th>
<th>PopSize</th>
<th>Preselect</th>
<th>Traffic</th>
<th>Convergence</th>
<th>Improvement over ES/HC</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>100</td>
<td>10%</td>
<td>1-100</td>
<td>25%</td>
<td>Less than 10%</td>
<td>~3s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10%</td>
<td>1-150</td>
<td>70%</td>
<td>Less than 10%</td>
<td>~3s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5%</td>
<td>1-150</td>
<td>20%</td>
<td>Less than 10%</td>
<td>~3s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1%</td>
<td>1-150</td>
<td>0%</td>
<td>Less than 10%</td>
<td>~3s</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>1%</td>
<td>1-150</td>
<td>40%</td>
<td>10%</td>
<td>~5s</td>
</tr>
<tr>
<td>27</td>
<td>100</td>
<td>10%</td>
<td>1-100</td>
<td>10%</td>
<td>Less than 10%</td>
<td>~1m</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>10%</td>
<td>1-150</td>
<td>40%</td>
<td>Less than 10%</td>
<td>~1m</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>10%</td>
<td>1-150</td>
<td>20%</td>
<td>10%</td>
<td>~3m</td>
</tr>
<tr>
<td>81</td>
<td>300</td>
<td>10%</td>
<td>1-100</td>
<td>0%</td>
<td>10%</td>
<td>~10m</td>
</tr>
</tbody>
</table>

Table 5-3. Results Using OCX

The results are far from optimal. Most of the time (about 60%) the GA fails to even converge. Even in the best case scenario (Preselect = 10%), it still has only 70% chance to succeed. And such highly biased selection could cause early convergence and disrupt the natural evolution of the algorithm as well. However, whether the GA converges or not, the best fitness in the final generation is always about 10% higher than the best fitness found by ES and HC. In such sense, OCX does show some improvement, but not much. A number of parameters such as population size, threshold value, traffic matrix and maximum number of generations, have been varied during experiments. But none of them leads to any significant improvements.
There are a number of factors contributing to the poor performance of the OCX strategy. First, it does not use any information about the nature of the mapping problem and does not exploit the information in the structure of the chromosomes. The original idea of OCX is to protect sub-tours in TSP, which are irrelevant to this mapping problem. Therefore, the crossover process appears to be random and does not attempt to preserve good genes. Chromosome contention is also found to be a problem, when using OCX. Chromosome contention is a phenomenon where different best chromosomes contest with each other and crossover between two different best chromosomes does not create a good offspring. This eventually results in a right-tail distribution [Gol89], where over 70% of the fitness values of the chromosomes stay in a narrow area, which is far from the best fitness. Selection has no pressure and the result after mating is random. In other words, the algorithm stagnates and never converges.

There are two ways addressing this problem:

i) Insert good chromosomes which are significantly better than the others into the initial population and

ii) Employ more meaningful crossover strategies.

Both of these approaches are discussed in the following sections. Nevertheless, based on the above thorough experiments, it is concluded that OCX is not suitable for the target problem.

As shown in Table 5–2, ES and HC perform poorly (only 10% improvement over the initial population), it is meaningless to compare the performance of the proposed algorithm with theirs. It is also inappropriate to compare with others because they either focus on irregular topologies (like HLDA) or can only deal with small size
problems (like MILP). But the proposed approach is based on regular topology and it is part of the purpose of this research to examine its response to large size problems. In addition to comparison with other algorithms, an algorithm can also be evaluated based on the global optimum. Therefore, the artificial traffic matrix is introduced. As explained early in Section 5.1, its minimal congestion can be easily determined. From now on, all tests are based on the artificial traffic matrix and the complement operator is also put into practice.

5.3 Sub-graph and Cluster Crossover Strategies

This section studies the effectiveness of SCX and CCX as the crossover strategies. As discussed in Chapter 4, these strategies make use of information about the target application area and they are expected to perform better than OCX. Table 5–4 and Table 5–5 show the performance of these two strategies. In order to provide and maintain good diversity inside the population, a larger population size (300) is used. The minimal and maximal numbers of generations are 200 and 400.

<table>
<thead>
<tr>
<th># of Nodes</th>
<th>PopSize</th>
<th>Preselect</th>
<th>PR</th>
<th>CR</th>
<th>Converge</th>
<th>Performance</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Best</td>
<td>Second</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>10%</td>
<td>33%</td>
<td>4%</td>
<td>95%</td>
<td>40%</td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5%</td>
<td>33%</td>
<td>4%</td>
<td>100%</td>
<td>35%</td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1%</td>
<td>33%</td>
<td>10%</td>
<td>40%</td>
<td>30%</td>
<td>60%</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>1%</td>
<td>33%</td>
<td>10%</td>
<td>80%</td>
<td>70%</td>
<td>30%</td>
</tr>
</tbody>
</table>

Table 5–4. Test Result for SCX
<table>
<thead>
<tr>
<th># of Nodes</th>
<th>PopSize</th>
<th>Preselect</th>
<th>PR</th>
<th>CR</th>
<th>Converge</th>
<th>Performance</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>100</td>
<td>1%</td>
<td>67%</td>
<td>4%</td>
<td>90%</td>
<td>50% 50% 100%</td>
<td>~3s</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>1%</td>
<td>67%</td>
<td>6%</td>
<td>85%</td>
<td>75% 25% 100%</td>
<td>~3s</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>1%</td>
<td>67%</td>
<td>10%</td>
<td>95%</td>
<td>90% 10% 100%</td>
<td>~5s</td>
</tr>
<tr>
<td>16</td>
<td>300</td>
<td>1%</td>
<td>50%</td>
<td>10^9</td>
<td>0%</td>
<td>0% 50% 50%</td>
<td>~20s</td>
</tr>
</tbody>
</table>

Table 5–5. Test Result for CCX

For a 9–node network, both strategies perform extremely well. CCX reaches convergence 95% of the time and is able to find the best (or one of the best) solutions at least 90% of the time. For SCX the corresponding values are 80% and 70% respectively. However, for a 16–node network (using the same setting as the 9–node CCX experiment) the program never converges. The problem is quickly identified: the rapid dropping of CR and PR.

As going from a 9–node network to a 16–node network, the search space increases from 9! to 16!. Using the same population size of 300 and the same maximum of 400 generation, CR actually drops from 10^-1 (300*400/9!) to 10^-9 (300*400/16!). To compensate, the population size is increased to 900 and the maximum number of generations to 1000 in the next experiment of RCX. As a result, CR increases to about 10^-7, which is still very low.

In a 3^2 nodes network each cluster consists of 5 to 7 nodes, whereas in a 4^2 network it has 7 to 9 nodes. So, as the network size increases from 9 to 16, the protection ratio for CCX drops from 66% (6/9) to 50% (8/16). This ratio keeps dropping as the network size continues to increase. This means CCX is not protecting enough of the
good genes in a chromosome. These issues are addressed in the next sections.

5.4 Random and Mixed Crossover

One factor affecting the performance of SCX and CCX is the rapid reduction in the protection ratio as the network size grows. In order to address this issue, two new crossover schemes are introduced: Random Crossover and Mixed Crossover. In RCX each gene in a chromosome has a 50% chance of being switched. The genes to be switched are selected randomly and RCX does not use any information about the particular application. RCX guarantees PR to be 50% regardless of the network size. The next crossover scheme, MCX, can be considered a combination of all proposed crossover strategies. It is expected to protect clusters (the building blocks) among chromosomes and yet maintain a certain level of diversity to avoid early convergence. Also the PR ratio of MCX is not affected much by the changing of the network size.

The results for these two crossover strategies are given below in Table 5-6 and 5-7.

For both experiments, the preselection value is 1%. When the population size is 900, the minimal and maximal numbers of generations are 200 and 1000; for a population size of 3000, they are increased to 400 and 10000.

<table>
<thead>
<tr>
<th># of Nodes</th>
<th>PopSize</th>
<th>PR</th>
<th>CR</th>
<th>Convergence</th>
<th>Performance</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Best</td>
<td>30%</td>
</tr>
<tr>
<td>16</td>
<td>900</td>
<td>50%</td>
<td>$10^{-7}$</td>
<td>100%</td>
<td>0%</td>
<td>30%</td>
</tr>
<tr>
<td></td>
<td>3000</td>
<td>50%</td>
<td>$10^{-6}$</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td>27</td>
<td>3000</td>
<td>50%</td>
<td>$10^{-27}$</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 5-6. Test Result for RCX
<table>
<thead>
<tr>
<th># of Nodes</th>
<th>PopSize</th>
<th>PR</th>
<th>Convergence</th>
<th>Performance</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Best</td>
<td>Second</td>
</tr>
<tr>
<td>9</td>
<td>200</td>
<td>50%</td>
<td>45%</td>
<td>80%</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>50%</td>
<td>75%</td>
<td>85%</td>
<td>15%</td>
</tr>
</tbody>
</table>

Table 5–7. Test Result for MCX

In both of the above experiments the algorithm never finds the best solution for a 16-node network or a 27-node network. The main reason for the results given above is that, for larger networks, the search space becomes so large that no crossover strategy with such a small population can cover a reasonable amount of the search space within such limited number of generations. It is very difficult for the algorithm to find the proper direction and the search is pretty much random before settling down in one direction. When the coverage ratio is higher, the algorithm has a more meaningful search and crossover strategies that exploit additional information are expect to lead to better solutions. This is exactly what happens when comparing OCX with SCX and CCX for a 9-node network. This effect can also be seen, to a certain degree, in the 16-node network experiments. In this case, when the coverage ratio is increased, by increasing the population size from 900 to 3000, the algorithm is always able to find the second best solution.

It is also interesting to note that MCX does not perform as well as what is expected. It does not seem to offer significant advantages over SCX and CCX. And a more concerning fact is its low convergence: it does not have the power to hold good genes together. For these reasons, this thesis discards the possibility of further study on MCX.
5.5 Inserting Good Genes into Initial Population

It is clear from the results presented in the preceding sections that, for larger networks, irrespective of the crossover strategy used, it is almost impossible to get satisfactory results if the algorithm starts with a set of randomly generated chromosomes in the initial population. If the initial population does not have some good chromosomes to focus and guide the search, chromosome contention is very high and it takes a lot of time and effort to settle down for the correct search direction. Therefore, the initial population needs to be chosen in such a way that it contains at least one (and possibly several) chromosome(s), which contain some good genes. Obviously, the question arises, if the optimal solution is unknown, how does the algorithm know which chromosomes are good and how does it generate such chromosomes? One possible approach is to use heuristic algorithms, such as those suggested in [RS98, ZA95], to generate some initial solutions (chromosomes), which are likely to contain good genes.

It is the purposes of this investigation to study how often the algorithm is able to find the global optimum, if some good genes are inserted into the initial population. To do this, the global optimum has to be known in advance. So the artificial traffic matrix is used again in the next set of experiments. For such a traffic matrix, it is easy to calculate one chromosome corresponding to the best solution. This thesis has taken this chromosome, modified it slightly and put it into the initial population. Table 5–8 shows that there is a significant improvement when using an improved initial population. For a 16–node (27–node) network the algorithm has a 75% (45%) chance of finding the best or second best solution. Note that for all tests in Table 5–8, the preselection value is 1%; the population size, minimal and maximal numbers of generations are 300, 200 and 400 respectively for the 9–node network; for the 16–
node network they are 900, 200 and 1000; and 3000, 400 and 10000 for the 27-node network.

<table>
<thead>
<tr>
<th># of Nodes</th>
<th>CX</th>
<th>Refill Frequency</th>
<th>PR</th>
<th>CR</th>
<th>Converge</th>
<th>Performance</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>CCX</td>
<td>First G</td>
<td>67%</td>
<td>10%</td>
<td>95%</td>
<td>90% 10% 100%</td>
<td>~5s</td>
</tr>
<tr>
<td></td>
<td>RCX</td>
<td>First G</td>
<td>50%</td>
<td>10⁻¹⁷</td>
<td>100%</td>
<td>45% 30% 75%</td>
<td>~4m</td>
</tr>
<tr>
<td></td>
<td>RCX</td>
<td>Every G</td>
<td>50%</td>
<td>10⁻¹⁷</td>
<td>100%</td>
<td>100% 0% 100%</td>
<td>~4m</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RCX</td>
<td>First G</td>
<td>50%</td>
<td>10⁻²⁷</td>
<td>100%</td>
<td>0% 45% 45%</td>
<td>~20m</td>
</tr>
<tr>
<td></td>
<td>RCX</td>
<td>Every 10 G</td>
<td>50%</td>
<td>10⁻²⁷</td>
<td>100%</td>
<td>0% 45% 45%</td>
<td>~20m</td>
</tr>
<tr>
<td></td>
<td>RCX</td>
<td>Every G</td>
<td>50%</td>
<td>10⁻²⁷</td>
<td>100%</td>
<td>100% 0% 100%</td>
<td>~20m</td>
</tr>
</tbody>
</table>

Table 5–8. Test Result for Good–gene Effect

Further analysis of the raw data reveals other interesting results. When the GA successfully finds the best solution, it always happens within the first 100 generations i.e. it either finds the best solution very fast or never. Moreover, for larger networks, when a good solution is found, it is always contains the original good genes inserted in the first generation. In other words, if the good genes are not destroyed in the first couple of generations, they dominate the population very quickly. Otherwise, they disappear forever. Therefore, the performance of the algorithm (for larger networks) depends heavily on the survival of the good genes.

To verify this theory, something called refilling is introduced. The algorithm puts the good genes back into the population every 10 generations. However, refilling every 10 generations does not show any significant improvement. So the refilling frequency is increased to every generation. In this case, the GA is always able to find the best
solution.

However, the test results on good-gene effect for the 9-node network are completely different. The program maintains 90% chance of finding the global optimum. But the solution does not have any correlation with the genes inserted into the initial population. It may find an optimal solution that is quite different. The effect of the good genes is not significant at all. The reason is that the CR is so high \(10^{-1}\) for a 9-node network that the algorithm is able to find the best solution even without the help of good genes. This indicates that if the population size is sufficiently large, i.e., CR is correspondingly high, the algorithm might be able to compensate for the "good-gene" effect. The experiments for the 16-node problem tend to verify this theory. For a 16-node network with good genes, population size 900 and a maximum of 1000 generations, the algorithm has 45% chance of getting the best, and 30% chance of getting the second best solution. Without the good genes and with an increased population size of 3000 and a maximal number of generations of 10000, the algorithm can not find the best but always come up with the second best. The average performance is in fact improved.

5.6 Conclusion

In this chapter, a series of experiments was conducted to test the proposed algorithm. The result is far more than satisfactory, which can be summarized as follows:

§ For a 9-node network, the algorithm can always find the best solution with or without good genes;
§ For a 16-node network, with the help of good genes the algorithm has 45% chance to find the best plus another 30% to find the second best; otherwise, 100% for the second best with a larger population size;

§ For a 27-node network, with good genes in the first generation, the algorithm has 45% chance to find the best or second best; with refilling every generation, it always finds the best; but without any help of good genes, it can only find the third best;

With the CR ratio decreasing from $10^{-1}$ to $10^{-7}$, and to almost 0, the above results are quite reasonable. A summary of the important test results is given in Table 5–9.
<table>
<thead>
<tr>
<th># of Nodes</th>
<th>OCX</th>
<th>SCX</th>
<th>CCX</th>
<th>RCX</th>
<th>ES / HC</th>
<th>CR (for GA only)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 (3²)</td>
<td>$PC = 70%$</td>
<td>$PC = 80%$</td>
<td>$PC = 95%$ Over 90% gets the best solution</td>
<td>Not tested</td>
<td>$PR = 75%$</td>
<td>300*150 / 9! = 10%</td>
<td>3–5 seconds</td>
</tr>
<tr>
<td></td>
<td>PR = 50%, but fluctuates</td>
<td>PR = 33%, fluctuates</td>
<td>$PR = 67%$, stable</td>
<td></td>
<td>$PI = 10%$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 (4¹)</td>
<td>Not tested</td>
<td>Not tested</td>
<td>$PC = 95%$ 50% for second best</td>
<td>$PC = 100%$ With good gene in first generation, 45% gets the best and 30% second best</td>
<td>$PR = 88%$</td>
<td>1000*10000 / 16! = 10⁻⁷</td>
<td>3–5 minutes</td>
</tr>
<tr>
<td></td>
<td>PR = 50%, but fluctuates</td>
<td>PR &lt; 30%</td>
<td>$PR = 50%$, stable</td>
<td></td>
<td>$PI &lt; 10%$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27 (3¹)</td>
<td>$PC = 40%$</td>
<td>Not tested</td>
<td>Not tested</td>
<td>$PC = 100%$ With good gene in first generation, 45% gets the best or second best</td>
<td>$PR = 93%$</td>
<td>3000*10000 / 27! = 10⁻²⁷</td>
<td>15–20 minutes</td>
</tr>
<tr>
<td></td>
<td>PR = 50%, but fluctuates</td>
<td>PR &lt; 25%</td>
<td>$PR = 25%$, stable</td>
<td></td>
<td>$PI &lt; 10%$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$ PC$ is the percentage of convergence; $PI$ is the percentage of fitness improvement over the first generation.

Table 5–9. Summary for Test Results
CHAPTER 6
Conclusions and Future Work

This chapter summarizes the contributions of this research and discusses the conclusions that have been reached. It also outlines some directions for future work and strategies for further improvement.

6.1 Conclusions

This thesis has attempted to solve a simplified version of the connectivity problem and flow assignment problem for logical topology design of optical networks. The search has been restricted to find a mapping with "minimal" congestion in a De Bruijin graph provided that the traffic matrix is given. A genetic algorithm was proposed to solve this problem and the Alternative Path Algorithm was introduced as the objective function. This research was also extended to a more general De Bruijin graph based topology which can accommodate an arbitrary number of nodes.

During this research, a series of experiments was conducted in a controlled environment. The results are intriguing and encouraging. For a 9-node network with an artificial traffic pattern, the algorithm can come up with the best solution in over 90% of the time, and surprisingly it only takes a couple seconds to achieve such a result; when the network size increases to 16, with the help of good genes, it still has
75% chance of getting the best or second best result; but for a 27-node network, the performance drops dramatically.

This thesis identified and analyzed how different factors including CX strategy, population size, number of generations, the good-gene effect, chromosome contention and dropping CR/PR phenomenon affect the performance of the GA. It also introduced a number of new crossover strategies like SCX, SCX and CCX and showed that they can be effective in guiding the GA to the best solution. The algorithm also shows the potential to deal with large size networks.

One of the most promising features of the proposed GA is its efficiency. For a 9-node network, it travels only 10% of the search space to reach the global optimum. Even for a 27-node network, it only takes approximately 20 minutes to find a reasonable solution. For even larger networks, it may be necessary to use Parallel Genetic Algorithm (PGA), which is discussed in Appendix D. Although conventional MILP and LP formulations can always find the best topology, it takes a great deal of time (couple hours) to achieve such a result, which narrows their applications for small networks only. On the contrary, the proposed algorithm is able to handle much larger networks because of its high efficiency: reasonable solution within reasonable amount of time. And such high efficiency plus the inherited flexibility of GA suggests that there is potential for collaboration with other algorithms to further refine the results.

The analysis of coverage ratio indicates that sub-optimal results can sometimes be expected simply because of the huge search space. This can be compensated, to some extent, by increasing population size and running more generations, i.e. higher computational cost. With the problem size continuing to increase, this cost might rise
exponentially. The introduction of PGA enhances the potential of the proposed GA to deal with large size networks.

In conclusion, based on what has been achieved so far, GA appears to be a very useful tool for the target application area. This research is part of an ongoing study to determine how a regular logical topology performs with respect to an irregular logical topology.

6.2 Future Work

This section outlines some related topics which merit further investigation and have the potential to significantly improve the algorithm.

1. Application of PGA: all data gathered so far points to the direction of PGA as the future. CR analysis suggests a larger population might be able to maintain the performance for large size networks. It means for a 27-node network the algorithm needs a $10^{10}$ population to run $10^{10}$ generations in order to maintain performance levels similar to those of a 16-node network. This type of processing is practically impossible for sequential GA to handle. "Divide and conquer" techniques like PGA are always very useful in such case. However, as one of the final frontiers in GA research, PGA has several unknown factors including migration time, migration rate and sub-population topology. A great deal of effort is required.

2. Permutation Schemata: Holland's binary schemata are not suitable for the proposed algorithm. New schemata based on permutation representation are needed. Proper schemata specification can provide better understanding on issues
like the CX working mechanism, distance function, chromosome contention, chromosome similarity, the good-gene effect, epistasis etc. Such solid theoretical foundation might also help find better CX strategies and stop criteria.

3. Improvements to CCX: CCX takes advantage of available auxiliary information and works very well for 9-node networks. The reason for its degrading performance with larger networks is the rapid dropping of its PR ratio. One solution is to protect more than one cluster for large networks. But the relationship between the number and position of protected clusters and the network size demands further investigation.

4. Multiple-objective: from the end users' perspective, short delay is more important than low congestion. Therefore, redefinition of the objective function to include delay is needed. This is no easy task because the well-known Pareto Optimality [NC93, WM93] in GAs does not work very well on some occasions, particularly when objectives are not clearly separated, or when they are not independent from each other like the case of this study. A new definition of optimality is needed.
References


10. [BV91], David L. Battle, Michael D. Vose, "Isomorphisms of Genetic
11. [Boo91], Lashon B. Booker, "Representing Attribute-Based Concepts in a Classifier System", in [Raw91], pp. 115 – 127.


20. [FM93], Stephanie Forrest, Melanie Mitchell, "What Makes a Problem Hard for a

21. [For85], S. Forrest, "Documentation for PRISONERS DILEMMA and NORMS Programs that Use the Genetic Algorithms", unpublished manuscript, University of Michigan, Ann Arbor, 1985.


39. [Muk97], B. Mukherjee, "Optical Communication Networks", McGraw Hill Text,
1997.


42. [NY99], Alioune Ngom, Svetlana N. Yanushkevich, "Neuro–Genetic Minimization of Multiple–Valued Logic Sum–of–Products Expressions", Computer Science Department, Lakehead University, 1999.


45. [RS95], R. Ramaswami, K. N. Sivarajan, "Routing and Wavelength Assignment in All–optical Networks", IEEE/ACM Transactions on Networking 1995; 3; 489–500.


60. [WM93], P. B. Wilson, M. D. Macleod, "Low Implementation Cost IIR Digital

61. [Wri91], Alden H. Wright, "Genetic Algorithms for Real Parameter Optimization", in [Raw91], pp. 205 – 220.

Appendix A Definitions

Allele: Each element value in a chromosome is called an allele.

Alternative Path: An alternative path from node $N_1$ to node $N_2$ is any path from $N_1$ to $N_2$. In a De Bruijn graph, there are infinite number of alternative paths (including paths with self loop) between any two nodes.

Artificial Traffic Matrix: It assumes in the best mapping, a node only communicates with its adjacent members. Its minimal congestion is easily calculable.

Centralized Traffic Matrix: It is a random traffic matrix except a small number of elements are significantly larger than the others.

Chromosome: An abstract, coded representation of the solution to the target problem.

Chromosome Contention: Several best chromosomes compete with each other and none can thrive, resulting in stagnation. It usually occurs in multiple-peak problems.

Convergence: A situation where individuals in a population become identical or similar to each other. It usually indicates that the GA has slowed to a point that it does not seem to find new, better solutions.

Coverage Ratio: It evaluates how much space a GA travels before finding a good result.

Crossover: A genetic operator which creates a new chromosome by combining genetic material from two parent chromosomes.

Crossover Point: The split position in crossover. There might be more than one depending on what kind of crossover is used.

Early Convergence: A population is converged to a local optimum before reaching the global one.
Elicit selection: For every generation, a certain number of good chromosomes in the old generation automatically enter the next generation without going through the selection and mating procedure.

Epistasis: The interaction between different genes in a chromosome. It is the degree to which the contribution to the fitness of one gene depends on the values of the others.

Evaluation: A procedure that calculates a chromosome's fitness value using the fitness function.

Exploitation: The process of using information gathered from previously visited points in the search space to determine which places might be profitable to visit next.

Exploration: The process of visiting entirely new regions of a search.

Fitness: A value assigned to an individual which indicates the performance of this individual for the target problem. In simulation, it also represents the probability of the individual being selected for reproduction.

Fitness Function: A function maps a chromosome (individual) to a fitness value.

Gene: Each element position in a chromosome is called a gene.

Generation: A generation is a population indexed by time.

Generational Reproduction: Replace the whole population for every generation.

Good-gene Effect: A GA's performance for large networks heavily depends on the survival of good genes.

Individual: A single chromosome in a population.

Logical Topology: It is a logical perspective of a physical topology based on wavelength assignment.

Mutation: A genetic operator which creates a new chromosome by making random alteration(s) to the value(s) of the gene(s).

Offspring: A chromosome generated by reproduction.

Physical Topology: It is a network physically connected by optic fibers.
Population: A group of chromosomes which mate with each during reproduction to conceive offspring.

Premature Convergence: The same as early convergence.

Principle of meaningful building block: A coding should create short, low-order schemata relevant to the underlying problem and relatively unrelated to "the schemata over other fixed positions".

Principle of minimal alphabets: Users should choose the smallest alphabet set that permits a "natural expression" of the problem.

Proportional selection: The probability of being selected for a chromosome is equal to the fitness of the chromosome divided by the sum of the fitness in the population.

Protection ratio: It is the percentage of nodes that do not change during a crossover.

Random Traffic Matrix: Each element is randomly generated, but its range is not from 0 to 1.

Refilling: It puts the good genes back to the population every certain number of generations in order to protect them.

Reproduction: A procedure to create new generation.

Schemata: Descriptions of chromosome templates with the character set of 1, 0 and *, the "don't care" symbol.

Selection: A mechanism that according to their fitness, two chromosomes are selected for mating.

Stagnation: The average fitness of the whole population does not change too much in generations.

Steady-State Reproduction: Replace only a few members for each generation.

Stop criteria: The are the conditions under which a GA should stop.

Uniform Traffic Matrix: Each element in the matrix is randomly generated, ranging from 0 to 1.
# Appendix B Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCX</td>
<td>Cluster Crossover</td>
</tr>
<tr>
<td>CR</td>
<td>Coverage Ratio</td>
</tr>
<tr>
<td>CX</td>
<td>Crossover</td>
</tr>
<tr>
<td>ES</td>
<td>Evolutionary Strategy</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>HC</td>
<td>Hill Climbing</td>
</tr>
<tr>
<td>HLDA</td>
<td>Heuristic Logical Design Algorithm</td>
</tr>
<tr>
<td>LP</td>
<td>Linear Programming</td>
</tr>
<tr>
<td>MCX</td>
<td>Mixed Crossover</td>
</tr>
<tr>
<td>MILP</td>
<td>Mixed Integer Linear Programming</td>
</tr>
<tr>
<td>OCX</td>
<td>Order Crossover</td>
</tr>
<tr>
<td>PC</td>
<td>Percentage of Convergence</td>
</tr>
<tr>
<td>PI</td>
<td>Percentage of Improvement</td>
</tr>
<tr>
<td>PR</td>
<td>Protection Ratio</td>
</tr>
<tr>
<td>RCX</td>
<td>Random Crossover</td>
</tr>
<tr>
<td>SCX</td>
<td>Sub-graph Crossover</td>
</tr>
<tr>
<td>SGA</td>
<td>Simple Genetic Algorithm</td>
</tr>
<tr>
<td>TSP</td>
<td>The Travelling Salesman Problem</td>
</tr>
<tr>
<td>WDM</td>
<td>Wavelength division multiplexing</td>
</tr>
<tr>
<td>WRON</td>
<td>Wavelength routed optical network</td>
</tr>
</tbody>
</table>
Appendix C Traditional Search Methods and Their Limitations

In order to find out why GAs are so attractive, this article takes a look at how other conventional optimization methods work and explores their limitations. Based on Goldberg's categorization [Gol89], there are three kinds of search methods: calculus-based, enumerative and random. And this review adds one more here: Randomized Search Methods.

1. Calculus-based Methods

A calculus-based method, also called gradient search, finds the local best and climbs the function in the "steepest permissible direction" [Gol89]. A well-known example of this method is Hill Climbing (HC), which moves upwards in a slope in hopes of finding the maximum. Although experimental results from the Bit-strings Problem [Dav91], the Multiprocessor Document Allocation Problem (MDAP) [JW94], and the "Royal Road" Function [MHF94] show that in some occasions, HC can outperform GA, people still criticize that calculus-based methods including HC are not robust for the following two reasons [Gol89, Wil98]. First of all, they work well at single-peak functions but not good at multiple-peak problems. For example, in Figure 1, the algorithm starts search from point A. It quickly finds point C and then stops because it can not find a higher/better point in the neighborhood of the current position C. Therefore, the algorithm is trapped in a local optimum and never finds the global one (point B in this case). The introductions of Stochastic Hill Climbing (SH) [JW94] and Stochastic Iterative Genetic Climbing (SIGC) [Bet57], the improved versions of HC,
partially overcome this problem. However, these methods have another more serious problem.

![Graph](image)

**Figure 1.** HC is easily trapped in a local optimum.

The second and the most severe drawback of calculus–based methods comes from their dependency upon the existence of derivatives, i.e., the deviations of the given functions must be continuous. This restrictive requirement is unrealistic in practice and makes them work only in a limited domain. Therefore, these methods are "insufficiently robust" [Gol89] in unintended area.

2. **Enumerative Methods**

The enumerative method was originally introduced by Bellman in 1957 [Bel57] and ultimately discounted because of its lack of efficiency. This algorithm searches every point in a finite space, one at a time, based on their objective function values. However, many practical spaces are too large for it to search one at a time. The appearance of Dynamic Programming (DP) [Bel67], a highly recommended enumerative technique, keeps people's hope alive. This approach can search more efficiently and has the ability to handle local optimum, but in exchange for higher computation cost. In addition, DP breaks down on complex problem of modern size
and suffers from this situation, well-known as "curse of dimensionality".

3. Random Search Methods

Random search algorithm has been a rising star since researchers recognized the shortcomings of calculus-based and enumerative approaches. It is particularly popular for difficult problems where no other viable methods are available. Random-walk and save-the-best are the major strategies of this method, which is like looking for a needle in the ocean and obviously inefficient. In the long run, random search is expected to perform no better than enumerative approach.

4. Randomized Search Methods

Randomized Search Methods are new search techniques that do not fall in the above categories. The main difference between random search and randomized search is that the latter uses random processes as tools to help guide its search. Goldberg pointed out that "randomized search does not necessarily imply directionless search" [Gol89]. Therefore, this approach is expected to perform better.

The most famous randomized search method is Simulated Annealing (SA). Many of the critics of SA come from its search-one-at-a-time and its waste of history information.
Appendix D Advanced Topics on GA

The basic principle of GA is quite simple. But when put into practice, there are other issues have to be taken care of. Some topics discussed here might really complicate the situation.

1. Schemata

Holland’s schemata [Hol75] are the foundation of GA’s design methodology. The notion itself is simple. It states that schemata are sets of strings that have one or more features (bit value in his case) in common. A schema is built by introducing a "don’t care" symbol, "*", into the gene pool \( \{0, 1\} \). For example, "*101*10" is a schema. In an instance (copy or example) of a schema, "*" could be either 1 or 0. Therefore, 1101010 and 0101010 are both the instances of schema "*101*10". A schema contains \( 2^r \) instances where \( r \) is the number of "*" in the schema.

The order of a schema \( o(H) \) is the number of the fixed positions (total number of 1’s and 0’s ) presented in the schema. In the example above, the order is 5 (symbolically, \( o(*101*10) = 5 \)). The defining length of a schema, denoted by \( \&(H) \), is the distance between the first and last fixed positions. For instance, \( \&(101*10) = 7 - 2 = 5 \) because the first fixed position is 2 and the last one is 7.

Schemata and their properties are interesting devices for discussing and classifying string similarities. More than this, they provide basic means for analyzing the generation procedure —— the core of Genetic Algorithms.
Based on the above notion, Holland thoroughly and theoretically analyzed the effects of selection, crossover and mutation, and introduced the famous Schemata Theorem. Goldberg inherited this theory to study the effects of advanced genetic operators [GoI89]. The mathematical deduction detail is beyond the scope of this discussion. So just the equation and the conclusions are given below.

\[ M(H, t+1) \geq M(H, t) \times (f(H) / f_s) \times \left[ l_s - p_c \times \&c(H) / (l_s - 1) - a(H) \times p_m \right] \]

Where:

- \( M(H, t+1) \): for schema \( H \), the expected number of instances in generation \( t+1 \);
- \( M(H, t) \): for schema \( H \), the expected number of instances in generation \( t \);
- \( f(H) \): average fitness of the strings representing schema \( H \) in time \( t \);
- \( f_s \): average fitness of the entire population in time \( t \);
- \( p_c \): probability of crossover;
- \( \&c(H) \): defining length of \( H \);
- \( l_s \): the string length;
- \( a(H) \): order of schema \( H \);
- \( p_m \): probability of mutation;

The equation calculates the expected number of instances for schema \( H \) received in the next generation under proportional selection, crossover, and mutation. The implication of this equation is far from obvious:

§ "Above average" (in terms of fitness) schema receives an exponentially increasing number of strings in the subsequent generations due to selection.

§ The survival probability of a schema during crossover heavily depends on its defining length. The shorter the length, the higher the survival chance.

§ Destruction due to mutation is affected by the order of the schema. The lower the
order, the higher the survival chance.

$\S$ A high average fitness value is not sufficient for a high growth rate for a schema.

According to the above findings, Holland concluded the building block hypothesis:
"Short length, low order, above average schemata receive exponentially increasing trials in the subsequent generations of a GA" [Hol75]. He called these schemata building blocks. He pointed out that these building blocks represented the best partial solutions of past samplings. Instead of building high-performance strings by trying every conceivable combination, GA should work on building blocks. Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, an effective GA seeks near optimal performance through the construction and combination of short length, low order, highly fit schemata, or building blocks. Holland also approximated that for a population with size of $N$, the growth rate of the implicitly parallel processing is $O(N^2)$.

The building block hypothesis reveals the power source of GA, establishes the guidance for coding [Gol89], helps define effective crossover and selection scheme [TG94], and is even used to analyze the controversial GA-hardness problem [MHF94, FM93, Fos95]. Based on the Schemata Theorem and this hypothesis, Goldberg and Ded concluded that the time required by a GA to converge was $O(N \log N)$, where $N$ was the population size [Gol89, GD91].

However, Holland's notion merely focuses on the analysis of GA performance and is totally based on binary representation. With the rapid development of GA applications, this is no longer the case: a more general form of schema is needed [BV91]. Therefore, several researchers extended Holland's schemata to study more
complicated GA issues like permutation, GA-hardness and deception. For example, Bethke used Walsh–Schema Transform to study the static aspect of GA [Bet81], while Bridge and Goldberg proposed Nonuniform Walsh–Schema Transform [BG91] to explore GA's dynamics. And Tanes introduced Tanes Function [Tan89] to investigate GA deception. Other schemata includes "Royal Road" [MFH91], real parameter [Wri91] and Hyperplane Transforms [FM93, BG91].

2. Parallel Genetic Algorithms

As pointed out by Park and Carter [PC95], with increasing problem size, GA's performance decreases "ungracefully". This is partially because with bigger problem size, GA needs larger population size to maintain a pool of good genes; and it also needs a longer period (more generations) to converge. As a result, a higher computational cost is observed. One good solution to this problem is to divide a large population into small groups, called subpopulations, and let them compute independently and cooperatively in a parallel environment. They are independent because each subpopulation applies a separated GA internally without concerns about others; they are cooperative because from time to time they will exchange information to decide the direction of migration.

Ironically, GA, as a highly parallel algorithm, is implemented serially in most of its applications. It is not until late 80s that people paid much attention to Parallel Genetic Algorithm (PGA). Actually the history of PGA can be traced back to 1962 when Holland recognized the parallel nature of reproductive paradigm and discussed how to map reproductive plans to an iterative circuit computer [Hol62]. In 1981 Grefenstette [Gre81] proposed four implementations of PGAs, two of them known as global
parallelization and coarse grained GA.

A global parallelization GA has only one population. The evaluation of individuals and the application of genetic operators are assigned to each of the processors and done in parallel. The semantics of the operators are not changed. This method is easy to implement and the performance is proportional to the number of processors available. On the other hand, The coarse grained GA divides a population into a few subpopulations and keeps them isolated from each other. Mating is done only inside the subpopulation. But individuals in one subpopulation can migrate to another through a migration operator.

The new generation of PGA was fine grained PGA introduced by Robertson [Rob87], which partitions the population into a large number of very small subpopulations. The ideal case is to have just one individual for every processing element available. This model calls for massively parallel computers.

Although PGA promises a lot in terms of speedups and the improvement of search quality, Dr. Eric pointed out several problems that are not understood at all [Can95]:

§ Migration timescale: what is the proper time interval for migration?
§ Migration rate: how many individuals should migrate from one subpopulation to another?
§ Topology: what is the best way to connect the subpopulations?

3. Epistasis

A gene's contribution to the overall fitness of an individual may be conditioned on the
values/positions of other genes. Such gene is called epistatic. In nature, such co-
dependency (or correlationhip) among genes occurs on a regular basis. For example,
bats have a gene that gives them keen hearing and another one to make high pitched
chirps. Neither of these genes alone would improve a bat's fitness. But together they
form a sonar system and have a major impact on its fitness.

In practice, the amount of epistasis varies from none to severe. For example, flipping
a coin 8 times does not have any epistasis. Each flipping does not interfere with
others. On the hand, problems like scheduling and planing have very severe epistasis,
where the availability of a resource is dependent on the other schedules. An example
with moderate epistasis would be the plateau function, in which case genes have to
interact with each to reach the global optimum where they must all be "1".

Davis finds out that conventional techniques like Hill Climbing (HC) work very well
for low to moderate epistatic problems; but when such dependency increases to
severe, GA performs a lot better than the others [Dav91]. Such finding contradicts
with the building block hypothesis which requires low epistasis for a successful GA.
Michelle and Frost introduce "royal road" function to investigate this phenomena
[FM93, MFH91]. However, whether such correlation among genes is the cause of
GA—hardness and to what degree they "contribute" to such cause is still a mystery.

The common knowledge is to avoid or minimize epistasis. One solution is to change
the chromosome representation. Goldberg points out that as a principle, a coding
should be relatively unrelated to "the schemata over other fixed positions" [Gol89].
Vose and Liepins demonstrate that it is possible to code any problem in such a way
that it has little or no epistasis [VL91]. But the work of such transformation might be
more than that of solving the problem itself. Another alternative is to customize crossover and mutation operators for the specific problem, which is done in this research.

In the proposed mapping problem (Section 1.3), the heuristic is that if two nodes have heavy traffic, they should be assigned to be as close to each other as possible. It means that the contribution of one gene in a chromosome heavily depends on the positions of the others. In other words, correlation is very strong in this problem. Therefore, such phenomenon has to be dealt with. It could be crucial to the success of this research.
Vita Auctoris

Name: Zhiyong Michael Liu
Place of Birth: Shunde, Guangdong, P. R. China
Year of Birth: 1974

Education:
- Shunde No. 1 High School, Shunde, Guangdong, P. R. China
  1984 – 1992
- Zhejiang University, Hangzhou, P. R. China
- University of Windsor, Windsor, Ontario, Canada

Related Experience:
- System Administrator
  Huanan Air Conditioning Ltd., P. R. China
  1996 – 1997
- Technical Manager
  Goldenwarre Computer Co., P. R. China
  1997 – 1998