



THE HONG KONG  
POLYTECHNIC UNIVERSITY

香港理工大學

Pao Yue-kong Library  
包玉剛圖書館

---

## Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

**By reading and using the thesis, the reader understands and agrees to the following terms:**

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.
2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact [lbsys@polyu.edu.hk](mailto:lbsys@polyu.edu.hk) providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.

**HONG KONG POLYTECHNIC UNIVERSITY**  
**DEPARTMENT OF COMPUTING**

**A Graph Based Evolutionary Algorithm**

**Wong Shing Yue Samuel**

**A thesis submitted in the partial fulfillment of the  
requirements for the degree of Doctor of Philosophy**

**October 2008**

## **CERTIFICATE OF ORIGINALITY**

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

Wong Shing Yue Samuel

Department of Computing

Hong Kong Polytechnic University

October 2008

---

# Abstract

---

A number of different Evolutionary Algorithms (EAs) have been developed to evolve different kinds of graph structures. The most common being those that evolve Artificial Neural Network (ANN) architectures and those that evolve trees. In other words, these EAs can only be used to evolve specific graph topologies and they cannot be easily adapted to evolve graphs in general. Given that many data structures can be represented as graphs, a general EA that can tackle graphs with no specialized topology can have many useful real world applications. Towards this goal, this thesis proposes a general EA for graphs called EvoGraph. EvoGraph can be used to evolve all kinds of graphs by encoding them in adjacency matrices. Like other heuristic search algorithms, evolutionary search in a space of graphs also faces the challenge of a tremendously expanded search space when there is an increase in the number of nodes in a graph. Though this can be mitigated by increasing the population size to provide a larger variety of building blocks for the search, this increase in population size is limited, however, by constraints in computational resources. To address this problem, it should be noted that previous work has shown that uniform crossover in Genetic Algorithm (GA) could be rather efficient for heuristic searches in a large search space with a relatively small population. This operator ensures that all alleles have the same chance of being swapped during crossover and it also ensures that a relatively high degree of disruption be introduced to ensure the generation of novel chromosomes. Based on such a principle,

EvoGraph's reproduction operators are also designed to resemble uniform crossover and mutation crossover in linear-string GA. The application of a single crossover operator in EvoGraph can achieve the same effect of more than one reproduction operation in a Genetic Programming (GP) and Evolutionary Programming (EP). EvoGraph can be shown to be very effective at various tasks involving the evolution of graphs in general and trees and ANN architectures in particular. EvoGraph is applied to solve a wide range of graph based heuristic problems considered in this thesis. They include architectural topology design, space frame design, creative art painting, molecular design and peer-to-peer overlay network design.

---

## Publications Arising From This Thesis

---

“EvoGraph: An evolutionary algorithm for graphs” submitted to *IEEE Transactions on Evolutionary Computation* for review. (Chapters 2 to 4 of this thesis refers)

“EvoArch: an evolutionary algorithm for architectural layout design” accepted for publication by *Elsevier Journal of Computer-Aided Design* for review on 20<sup>th</sup> April 2009. (Chapters 5 of this thesis refers)

“EvoMD: an algorithm for evolutionary molecular design” submitted to *IEEE Transactions on Computational Biology and Bioinformatics* for review. (Chapters 8 of this thesis refers)

“EvoNet: a graph evolutionary algorithm for balancing minimum spanning tree and shortest path tree” submitted to *IEEE-ACM Transactions on Networking* for review. (Chapter 9 of this thesis refers)

“Method for Automatic Generation of Optimal Space Frame” publication of patent application at *United States Patent and Trademark Office*, Publication No. US-2009-0073160-A1, 19<sup>th</sup> March 2009.

---

## Acknowledgement

---

The research that has one into this thesis has been thoroughly enjoyable. That enjoyment is largely a result of the interaction that I had with my supervisor, Dr. Keith Chan. I feel privileged to have worked with him and other staffs, Ms. Cherry Chan, and Ms. Mui Tai in the Department of Computing in Hong Kong Polytechnic University. To each of them I owe a great debt of gratitude for their patience, inspiration and friendship.

I would also like to thank Dr. Vincent Ng who guided me through the puzzles in computer codes when I occasionally drop into his room in the early days of my research. At that time I was working as a project associate in the Department. The Department has provided an excellent environment for my research. Without this rich environment I doubt that many of my ideas would have been able to come to fruition.

---

## List of Figures

---

**Figure 1:** Undirected graph encoding example

**Figure 2:** Directed graph encoding example

**Figure 3:** Tree encoding examples

**Figure 4:** General form of adjacency matrix for encoding ANN

**Figure 5:** ANN encoding example

**Figure 6:** Demonstration on Spanning Tree Generation

**Figure 7:** An example of a Graph Adjacency Matrix with Spanning Tree Highlight

**Figure 8:** Demonstration of Random Crossover of two Graphs

**Figure 9:** Demonstration of Random Crossover of two Trees

**Figure 10:** Standard GP Crossover to produce the same result as Random Crossover in Figure 9.

**Figure 11:** Random Crossover of ANN

**Figure 12:** Steps on evolving ANN using EP

**Figure 13:** The Number-of-Edge Mutation operator illustrated

**Figure 14:** The Number-of-Node operator illustrated

**Figure 15:** Permutation invariant of adjacency matrix illustrated

**Figure 16:** Number of connected graph topologies in w.r.t. number of nodes

**Figure 17:** Target DS-graphs to be evolved with their corresponding spectrums

**Figure 18:** Number of evolution operators in each generation of evolution applied in the experiments

**Figure 19:** Maximum fitness of resulting graphs generated by the experiments

**Figure 20:** Generation number on reaching maximum fitness of graphs evolved in the experiments

**Figure 21:** Adjacency Preference Matrix

**Figure 22:** The functional area floor plan of a house

**Figure 23:** Graph representation of the floor plan



**Figure 24:** Dual Graph (dotted lines and triangular nodes) derived from the graph in Figure 23

**Figure 25:** Functional Spaces enclosed by edges of Dual Graph (solid line and circular nodes)

**Figure 26:** The Node-Label Mutation operator illustrated

**Figure 27:** The Swap-Node mutation operator illustrated

**Figure 28:** Relation between Cost and Adjacency Preference Scale (APS)

**Figure 29:** Relation between number of nodes and converging generation

**Figure 30:** Histogram of experimental results

**Figure 31:** Cost efficiency vs relative APS ratio of property D to F

**Figure 32:** Adjacency matrix with APS generated by experiment 5

**Figure 33:** Optimal Architectural Space Topology generated by experiment 5 and corresponding floor plans

**Figure 34:** Space Frame Module repetition illustrated

**Figure 35:** Example of Cubical Symmetric Space Frame Module divided into subframes which are mirror images to each other

**Figure 36:** Example of a Pyramidal Symmetric Space Frame Module with pentagon base divided into subframes which are mirror images to each other

**Figure 37:** Examples on tetrahedron with different geometric regularities

**Figure 38:** Encoding three dimensional coordinates in linear chromosome

**Figure 39:** Hybrid algorithm process for EvoGraph and GA

**Figure 40:** Regular space frame modules generated by experiments

**Figure 41:** Module B with 13 edges

**Figure 42:** Violation of Mondrian subdivision rule

**Figure 43:** Example on tree encoding of Mondrian painting evolution

**Figure 44:** Comparing vertical and pinwheel subdivision of a square on the directional bias

**Figure 45:** Original Mondrian Painting ‘Composition with Red, Blue, Yellow’

**Figure 46(a):** Tree encoding on evolution of ‘Composition with Red, Blue, Yellow’

**Figure 46(b):** Attributed adjacency matrix for the tree in Figure 46(a)

**Figure 47:** Regression curve of maximum fitness of experiment sets w.r.t. crossover mix

- Figure 48:** Regression curve of average maximum fitness of the 5 experiment sets w.r.t. crossover mix
- Figure 49:** Regression curve of average maximum fitness of the 11 crossover mixes w.r.t. the 5 experiment sets in the order of increasing complexity of search
- Figure 50:** Regression curve of the converging generation of experiment sets w.r.t. crossover mix
- Figure 51:** Regression curve of average converging generation of the 5 experiment sets w.r.t. crossover mix
- Figure 52:** Regression curve of average converging generation of the 11 crossover mixes w.r.t. the 5 experiment sets in the order of increasing complexity of search
- Figure 53:** Basic steps of CAMD
- Figure 53:** Aspirin molecular graph and symmetric molecular matrix
- Figure 54:** Example of molecular graph generated by initialization
- Figure 55:** Molecular graph and adjacency matrices of aspirin and tylenol
- Figure 56:** Random cut line imposed on molecular matrices of aspirin and tylenol
- Figure 57:** Swapping of subgraphs created by the cut and deletion of invalid edges
- Figure 58:** Embed spanning tree at random to each offspring graph after swapping.
- Figure 59:** The Number-of-Node mutation operator illustrated
- Figure 60:** The Number-of-Edge mutation operator illustrated
- Figure 61:** The Swap-Node mutation operator illustrated
- Figure 62:** Comparison of 2 different overlay networks on the same underlay network
- Figure 63:** Comparison of costs of flow between of MST and SPT over the same underlay network
- Figure 64:** Two AMTCTs atop the underlay network in Figure 63(a) with corresponding costs of flow
- Figure 65:** Cost analysis of AMTCT1 and AMTCT2
- Figure 66:** Swapping of subtrees in GP producing invalid solutions for optimal routes
- Figure 67:** Encoding overlay tree over an underlay network in a cost matrix
- Figure 68:** An overlay tree generated atop an underlay network
- Figure 69:** Parent trees  $T^{P1}$  and  $T^{P2}$

**Figure 70:** Swap nodes '4', '5' between  $T^{P1}$  and  $T^{P2}$  on the right hand side of the cut line and delete invalid edges.

**Figure 71:** Generation of children spanning tree  $T^{C1}$  from degenerate tree  $T^{PC21}$  after exchange of subtrees in crossover.

**Figure 72:** Network mutation demonstrated

**Figure 73:** Linear relation between number of nodes and converging generations on search of AMTCTs by EvoGraph

**Figure 74:** AMTCT searched by EvoGraph atop underlay network in Figure 74(a) and cost comparison with corresponding SPT, MST.

**Figure 75:** Relation between degree density and number of additional AMTCTs.

---

## List of Tables

---

- Table 1:** Properties of a tree in adjacency matrix
- Table 2(a):** The function space description corresponding to the labels
- Table 2(b):** The Relative Room Ratios
- Table 3:** The probabilities of each operator being selected for reproduction
- Table 4:** Summary of Experiment Result
- Table 5:** Relative APS of Experiments
- Table 6:** Cost efficiency in relation to relative APS ratio of property D to F
- Table 7:** Regularity of geometric properties in Figure 37(a),(b),(c)
- Table 8:** Summary of converging generations and corresponding angle entropies and length entropies of space frame modules
- Table 9:** Symbols for tree encoding of Mondrian painting evolution
- Table 10:** Calculation of degree of bias of Figures 44(a) and (b)
- Table 11:** Attribute values calculation for Mondrian Painting in Figure 43
- Table 12:** Attribute values for ‘Composition with Red, Blue, Yellow’
- Table 13:** Design mix of GP and EvoGraph for Mondrian painting evolution
- Table 14:** Matrix identifying attributes to be captured in different experiments
- Table 15:** Converging fitness value
- Table 16:** Number of generations to reach convergence
- Table 17:** Summary of fitness functions and their target properties
- Table 18:** Illustration of fitness value calculation using  $f_1$
- Table 19:** First group experimental result using  $f_1$  as fitness function
- Table 20:** Evolving ibuprofen using  $f_2$  as fitness function
- Table 21:** Second group stage 1 experiments using fitness function  $f_3$
- Table 22:** second group stage 2 experiments using fitness function  $f_4$
- Table 23:** Second group stage 3 experiments using fitness function  $f_5$
- Table 24:** Degree of randomly generated underlay network

**Table 25:** Converging generation for experiments on different number of nodes

**Table 26:** Percentage cost saving results of AMTCTs on  $TC_{SPT}$  and  $TC_{MST}$  on each underlay network

**Table 27:** Fitness of AMTCTs generated in relation to degree density. The fitness of AMTCTs below the maximum fitness is underlined

---

## Table of Content

---

	Pages
Abstract	i-ii
Publications Arising From This Thesis	iii
Acknowledgement	iv
List of Figures	v-viii
List of Tables	ix-x
Table of Content	1-4
Chapter 1     Introduction	5-8
1.1 Background	
1.2 Objective of the Research and Contributions	
1.3 Outline of the Thesis	
Chapter 2     Literature Review on Graph-Based Evolutionary Algorithms	9-17
2.1 Evolution of Artificial Neural Network	
2.2 Genetic Programming	
2.3 Other Application Specific Graph Evolutionary Algorithms	
Chapter 3     EvoGraph Encoding Scheme for Different Graph Topologies	18-22
3.1 Encoding of Graphs in Adjacency Matrix	
3.2 Encoding of Tree	
3.2 Encoding of Artificial Neural Network	
Chapter 4     Reproduction Operators for EvoGraph	23-64
4.1 Generation of Spanning Tree	
4.2 Crossover of Graphs	
4.2.1 Random Crossover of Graphs	
4.2.2 Random Crossover of Trees	

	Pages
4.2.3 Comparison between Standard GP and Random Crossover of Trees in EvoGraph	
4.2.4 Random Crossover of Artificial Neural Networks	
4.2.5 Comparison between EP and EvoGraph on the Evolution of ANN Topology	
4.3 Mutation of Graphs	
4.3.1 Number-of-Edge Mutation	
4.3.2 Number-of-Node Mutation	
4.4 Mathematical Foundation for EvoGraph	
4.4.1 Exploration Verses Exploitation	
4.5 The EvoGraph Process	
4.6 Experiments on Comparison of Performance between EvoGraph Operators and Conventional Evolution Operators	
4.6.1 Fitness Function	
4.6.2 Experiments and Findings	
Chapter 5 Evolution on Architectural Space Topology	65-88
5.1 The problem of Architectural Space Topology Design	
5.2 Conversion Between Floor Plan and Graph Representing Architectural Space Topology	
5.3 Fitness Function	
5.4 Additional Mutation Operators	
5.4.1 Node-Label Mutation	
5.4.2 Swap-Node Mutation	
5.5 Experiments	
5.5.1 Initialization and EvoGraph Parameters Selection	
5.5.2 Experimental Results and Findings	
5.5.3 Optimal Architectural Space Topology Conversion to Floor Plans	

	Pages
Chapter 6      Evolution on Space Frame Topology and Geometry	89-106
6.1 Space Frame Design	
6.1.1 Symmetric Space Frame Module Topology Design	
6.1.2 Symmetric Geometric Properties and Dimensions Design	
6.1.3 Encoding Three Dimensional Coordinates of Space Frame Module Nodes in Linear Chromosome	
6.2 Fitness functions	
6.3 The Hybrid Evolutionary Algorithm for EvoGraph and GA	
6.4 Experimental Results	
Chapter 7      Tree Evolution on Art Creation	107-128
7.1 Quantification of Aesthetic Attributes of Mondrian Painting	
7.1.1 Directional Bias	
7.1.2 Evenness of Subdivision	
7.1.3 Color Distribution	
7.1.4 Granularity of Subdivision	
7.2 Experiments	
7.2.1 Fitness function	
7.2.2 Experiment Setup	
7.2.3 Evolution Parameters and the Evolution Process	
7.2.4 Fitness Analysis	
7.2.5 Converging generation analysis	
Chapter 8      Molecule Design	129-158
8.1 Existing Approaches for Computer Aided Molecular Design	
8.2 Encoding Molecular Design	
8.3 The Fitness Function	
8.3.1 Molecular Topology Descriptors	
8.3.2 Atom Adjacency Descriptors	
8.3.3 Fitness Functions for Evolution of Molecules	
8.3.4 An Illustrative Example in Drug Design	
8.4 Experiments on Evolution of Molecules	



	Pages
8.4.1 Initialization and Evolution Parameters	
8.4.2 Experimental Setup and Results	
Chapter 9    Peer-to-Peer Overlay Network Design	159-192
9.1 Previous Studies and the Problem	
9.2 Application of EvoGraph on Peer-to-Peer Overlay Network Evolution	
9.2.1 Generation of Tree on Overlay Network Atop an Underlay Network	
9.2.2 Crossover of Networks	
9.2.3 Mutation of Networks	
9.2.4 Fitness Function	
9.3 Experiments	
9.3.1 Initialization and Evolution Parameters	
9.3.2 Experimental Results	
Chapter 10    Research Conclusion and Future Directions	193-196
10.1 Research Conclusion and Contribution	
10.2 Limitations of the Research	
10.3 Future Directions of Research	
Appendix 1: Fitness Graphs on Best Performed Experiments on EvoGraph Operators and Conventional Evolution Operators	197-198
Appendix 2: Experimental Results on Architectural Space Topology Evolution	199-202
Appendix 3: Repetitive Space Frame Modules	203-208
Appendix 4: Mondrian Paintings Evolved by Tree Evolution	209
Appendix 5: Target Hydrogen Depleted Molecules and Corresponding Molecular Graphs	210-214
Appendix 6: Underlay Networks Cost Matrices with Approximate Minimum Total Cost Tree (AMTCT) and cost savings of each AMTCT	215-221
References	222-234

## Introduction

---

### 1.1 Background

Encoding of chromosomes is usually the first thing one need to decide when solving a problem with Evolutionary Algorithms (EA). This topic was introduced by Holland and his students around 1975 on genetic algorithm (GA) using binary string encoding. The first text book [1] on this topic enforced this presentation. It was only in early 1990s other types of EAs were proposed and studied. Many other encoding schemes, such as floating point presentation in chromosomes, permutation, matrices presentation of graphs by Michalewicz, and tree encoding by Koza, etc. [2][133] have been developed and used with much success when tackling different problems in different application areas. In addition to these popular encoding schemes, there have also been studies on cellular [12] or edge encoding [15], developed to encode tree-like structures. Similarly, work on genetic programming (GP) [2][3][13] and evolving Artificial Neural Networks (ANN)[4][14][16] has also been proposed on trees and neural networks architecture respectively. With a few exceptions in some domain-specific studies [17][18], there has not been much work done to develop EAs that can evolve graphs with different topologies.

### 1.2 Objective of the Research and Contributions

Given the diversity of real world problems that can be formulated in the form of graphs such as [5][6][17][18][131][132]. A number of different Evolutionary Algorithms (EAs) have been developed to evolve different kinds of graph structures. The most common being those that evolve Artificial Neural Network (ANN) architectures and those that evolve trees. In other words, these EAs can only be used to evolve specific graph

topologies and they cannot be easily adapted to evolve graphs in general. Given that many data structures can be represented as graphs, a general EA that can tackle graphs with no specialized topology can have many useful real world applications. Towards this goal, this thesis proposes a general EA called EvoGraph that can evolve connected graphs. EvoGraph encodes a connected graph (hereafter referred to as “graph”) in an adjacency matrix. It encodes connected graphs (hereafter referred to as “graph”) in adjacency matrices and makes use of a set of crossover and mutation operators designed to manipulate them. For an EA to effectively evolve graphs, it has to be able to handle a search space that would increase in size tremendously when the number of nodes of a graph, and hence the variety of topologies increases [69].

Though this can be mitigated by increasing the population size to provide a larger variety of building blocks for the searches, this increase in population size is often constrained by limited computational resources. To address this problem, previous work with linear-string GA makes use of the uniform crossover and mutation operators to efficiently evolve linear-string chromosomes in a large search space with modest-size populations. Such operators ensures that all alleles have the same chance of being swapped during crossover or changed during mutation and it also ensures that a relatively higher degree of “disruption” be introduced to avoid local optima and to ensure generation of novel chromosomes. Given such an advantage of the uniform crossover and mutation operators with the linear-string GAs, EvoGraph’s crossover and mutation operators are designed to operate like them. EvoGraph has several unique characteristics: (i) it encodes graphs in their adjacency matrices, (ii) it uses a novel crossover operator that can facilitate the exchange of characteristics between two graphs in a way equivalent

to the uniform-crossover operator for linear-string GA; (iii) it uses a set of novel mutation operators to facilitate the introduction of minor variations in topology in each graph to avoid trapping in suboptimal in the evolution process.

In addition to the introduction of a general EA for graphs without specific topologies, the random evolution operators of EvoGraph also introduce a new way of crossover and mutation for ANN and trees in addition to the conventionally adopted Evolutionary Programming (EP) and GP. We compare the advantages of EvoGraph operators over the conventional operators through hand simulations and experiments. After that, we apply EvoGraph to solve selected graph heuristic problems. The experimental results are promising.

### **1.3 Outline of the Thesis**

Different types of graph evolutionary algorithms are reviewed in Chapter 2. The most popular of which are ANN evolution and GP. This forms a reference for the development of a general evolutionary algorithm that is applicable to all types of graphs concerned. Chapter 3 introduces the new encoding scheme for different graph topologies into adjacency matrix. This is analogous to chromosome encoding in standard GA. Chapter 4 presents the different evolution operators of EvoGraph, *random crossover*, *Number-of-Node* mutation and *Number-of-Edge* mutation. The mathematical foundation of EvoGraph is also presented. Experiments on the performance of EvoGraph operators in comparison with conventional evolution operators on evolving graph without specific topologies, ANN and tree are illustrated. They demonstrate the advantage of EvoGraph over the conventional EAs. The above completes the formalism of EvoGraph. Chapters 5

to 9 illustrate different applications of EvoGraph. Through these applications, the potential of solving problems inherent in the existing graph evolutionary algorithms described in Chapter 2 is demonstrated. Chapter 10 concludes the findings and limitations of this research and indicates directions for future research.

## Literature Review on Graph Based Evolutionary Algorithms

---

The majority of current implementations of EAs descend from three strongly related but independently developed approaches: GAs, EPs and evolution strategies (ES). GA have been originally proposed as a general model of adaptive process by using recombination and mutation operators [135], but by far the largest application of the techniques is in the domain of optimization [136]. EP was originally designed to evolve finite state machines by using mutation alone [137], which transforms a sequence of input symbols into a sequence of output symbols. The performance of the finite state machine is measured on the basis of the machine's prediction capability. ES [138] was initially designed with the goal of solving evolution parameter optimization problem. It has been widely used in adaptive parameters tuning in dynamic evolution environment. In the 1990s, other types of EAs are developed. Such as, floating point presentation of chromosomes in addition to the original binary presentation in GA and evolution operators for drawing graphs proposed by Michalewicz [133]. Koza [2] also proposed GP for evolving computer programmes by encoding them in tree topologies. Recently, the most popular forms of graph evolutionary algorithm that attracts a lot of attention are the evolution of ANN and GP. These evolutionary algorithms are developed for the specific graph topologies, bipartite graph for ANN and tree for GP respectively. They are reviewed below to serve as a source of inspiration to develop a general encoding scheme and evolutionary algorithm for all graphs.

## 2.1 Evolution of Artificial Neural Network

A lot of work has been done to use EAs to evolve ANN architectures [4]. The goal is to find “optimal” or “near-optimal” architectures with respect to such criteria as the time of convergence of a learning process or the complexity of the network architecture.

A low level representation of a neural network is direct encoding and it is the most intuitive. Miller [26] maps the adjacency matrix of a neural network onto a binary string chromosome by concatenating the rows of the adjacency matrix. GA is then applied to evolve the network. Dodd [25] applies a GA to optimize a structured network for a pattern recognition problem classifying dolphin sounds. It is reported that a standard GA is able to find a network that is superior to any that can be created by hand. As the size of the network increases, the encoding of all details will result in a very long chromosome. The number of nodes of the network is fixed as the topology evolves. As the chromosome is formed by concatenating rows of the adjacency matrix into a linear string, the length of the chromosome has to be the square number of the dimension of the adjacency matrix. This square number constraint has to be maintained in order to enable successful decoding of the chromosome back into a graph. Hence, it does not favor variable length chromosome evolution.

A higher level representation is proposed by Harp [30] to swap layers of hidden nodes between neural networks. The network architecture is divided into a number of ‘blueprints’. Each ‘blueprint’ is described by several parameters like the number of nodes and its connection densities to some other ‘blueprints’. The ‘blueprints’ are mapped to a linear chromosome for evolution using a GA. A similar representation is proposed by

Mandischer [29], where for every layer receptive as well as projective connections are specified on the chromosome.

Other studies on encoding network layers in chromosomes include [32], which is an abstraction of the 2-dimensional grid structure of Parallel Distributed Genetic Programming (PDGP) proposed for edge recombination between nodes in different layers [34]. However, the abstraction of the 2-dimensional grid structure into a linear form involves numerous encoding and decoding rules. Note that these representations allow variable string lengths. The number of nodes is variable in the evolution process.

Another approach to achieve a desired ANN topology is to construct or modify a topology in incremental fashion. In view of the fact that the crossover operation in GA is not efficient in evolving both the weight and topology of an ANN at the same time, Yao and Liu [19] propose to use EPNet, which adopts only mutations in the process of evolution. Nodes are appended or deleted together with the modification of weights from one generation to another.

Another example of the use of an incremental approach is the evolution of projection neural network (PNN) [28]. The main difference between a PNN and an ANN is that PNN projects the original  $n$ -dimensional input vector onto the  $n+1$  dimensional vector that lies in the hyperplane of the original input vector and utilizes it as a new input vector. An operator for appending or deleting a node is required to change the dimension of the hyperplane that the node represents. In this regard, EP [24] on neural network is also applied. In [20], optimization of PNN by encoding the hidden node parameters such as weights and shaping factors in a link list data structure embedded in a chromosome is proposed. Ordered crossovers and mutations are applied. Another form of incremental



evolution of neural network by drawing analogy to biological growth is proposed by [33].

To allow dynamic expansion of topology of neural network during training, Opitz and Shavlik [31] proposed to allow exchange of hidden nodes and their attached links between two parent neural nets. Ordered crossover is adopted to achieve the required purpose and there is complete re-connection of links after nodes exchange to avoid the problem of disconnection. Some special cases such as neural network with tree topology are dynamically evolved by GP [14].

Another way to enable more dynamic change in topologies in the process of evolution is through indirect encoding. Indirect encoding generates graphs by reading instructions from genotypes in form of a chromosome [21] or a tree [12][15][22][130]. EA based optimization is then carried out at the genotype level. Other forms of indirect encoding involve evolution of graph generating rules such as that proposed by Kitano [23] or of encoding on a grammar based construction program in the genome proposed by Lindenmayer [27].

In summary, the different EAs and the numerous forms of encoding schemes and evolutionary algorithms described so far are designed to tackle the topological structure of ANN which consists of an input nodes layer, the hidden layers and the output nodes layer. Standard GA is typically used for directly encoded fixed node ANNs. However, direct encoding with standard GA is not suitable for variable node evolution because the length of chromosome is constrained by the square number of rows of the graph adjacency matrix. For variable node ANNs, ordered crossovers and mutations have to be used to maintain the basic structure of ANN throughout the evolution. Indirect encoding can provide a more dynamic change of topologies in the process of evolution but the

representation is less efficient because the evolution is not directly related to the network itself.

The above evolutionary algorithms for graphs or networks are ordered or constrained by the purposes they are designed to achieve. A general unbiased evolutionary algorithm for graphs free of topological constraints has not been developed. This thesis proposes a general evolutionary algorithm for graphs that is analogous to the standard GA [1]. The algorithm conducts crossovers and mutations on graph adjacency matrices while standard GA works on chromosomes. Given the general form of encoding in a graph adjacency matrix, it is simple to devise crossover and mutation operators to serve specific purposes.

## **2.2 Genetic Programming**

Genetic Programming (GP) has been widely adopted since its invention by Koza [2][3]. The crossover operator GP involves swapping of subtrees and this is analogous to one point crossover in standard GA except that it is carried out in a tree topology instead of a linear-string.

Much work has been done to develop crossover operator for GP. The most intuitive ones are those related to the study of the preservation of useful subtrees, or building blocks, during the evolutionary process. For example, Langdon [36] proposed ‘homologous crossover’ to improve the success of recombination by selecting subtrees deterministically, so that only subtrees with similar functions and topologies are exchanged. D’haeseleer [37] tried to assign indices to the nodes in preferred subtrees to enable exchange of matching branches instead of choosing subtrees at random in the crossover. In such circumstances, the context of the good population can be preserved.

However, constraints are imposed on the selection of nodes to exchange and this affects the efficiency of evolution. The good building blocks are also prohibited from spreading to other parts of the population.

Korkmaz and Üçoluk [38] calculate the fitness of subtrees during evolution and use the values to guide the recombination process so that subtrees with high performances are not destroyed during crossover. This algorithm learns from the evolution to determine the frequencies of good nodes in the subtrees and how they are distributed within the tree. Guidance for recombination is generated from this learning process to steer the crossover direction. It works well with subtrees that are not related to each other but not for trees having performances of subtrees that are dependent on each other.

Lones and Tyrell [39] proposed an algorithm to preserve the high performing subtrees and enable evolution of variable length solutions. The idea is to encode the nodes of a tree to behave like enzymes in chemical reactions. Each tree represents a program. Each node behaves like an enzyme that has its own function and selects its own interaction preferences. That is, each node selects which other nodes that it will react with. On the other hand, it has a representation that is conceived by other nodes. This is thus a mutual selection process. Unlike the crossover operation in a typical GP, in the proposed enzyme GP, a contiguous group of nodes is copied from one solution without removing any existing nodes. It is up to the other nodes within the tree to decide whether or not to use these new nodes.

There are studies on the design of appropriate crossover operators that tackles the problem of bloat, the building up of more and more redundant codes in a tree throughout the evolution process. Heywood [40][41][42][43] introduces ‘page base crossover’ to

minimize the code length. Each page in the code represents a tree which carries the same number of instructions. Since only subtrees having the same number of instructions are exchanged, the code lengths of the trees are kept under control. Therefore, less computational resource is used due to the limitation of code length of trees. ‘Depth dependent crossover’ [44] is also proposed to limit the code length. The probability of selection of node as a crossover point decreases from the top to the bottom of a tree. In solving some of the problems such as the *artificial ant* problem, the effectiveness of ‘depth dependent crossover’ is not exactly clear. Further exploration of the method is on going. Niimi and Tazaki [45] use reinforced learning through pruning redundant subtrees to control the tree sizes.

In view of the problem that there may be too much a demand on computational resources due to the increase in tree sizes throughout the process of evolution, studies are also conducted to reduce computational resources in crossover. In [46], Read’s linear code is used to encode tree structure in linear form. Though the presentation of a tree is simplified, the tree continues to increase in size as evolution proceeds. The problem of bloat has not been solved. Yanagiya [47] use binary decision diagrams to merge subgroups of nodes into one entity to facilitate evolution with a smaller number of nodes. This reduces storage requirements by sharing isomorphic subtrees among individuals, and saves computational power.

There are studies on crossover operations that adopt a more random approach when subtrees are exchanged. In [48][53], a ‘uniform crossover for GP’ is proposed. Based on it, two parents are being compared to identify, starting from the root node, the common regions where both of them have the same topologies. Then the interior nodes in the

common regions are swapped with probability 0.5. If a node belongs to the boundary of the common region, then the subtree below it is also swapped. Similar to the case of the uniform crossover operator in linear-string GA, such form of ‘uniform crossover in GP’ is less biased as it allows nodes in the common regions in the parent trees to be passed on to the offspring with a probability of 0.5. Unfortunately the identification of the common regions is very demanding in computational resources and this type of crossover relies on the existence of common regions between parent trees.

The different EAs described above adopt crossover operators that exchange subtrees between two parents. There have also been studies on crossover involving more than two parents, that is, multi-parent crossover [50]. Based on it, a number of subtrees are selected from a number of parents to exchange with each other. The process is analogous to chemical reaction equations where different chemical molecules react and recombine to form another set of different molecules. It is for this reason that this kind of crossover has also been referred to as ‘immune and the chemical crossover’. In this crossover process, the parents selected need to pass a recruitment test where they should exhibit higher fitness and similarity amongst themselves. Good zones of recruitment in the population are then identified. Each offspring is generated by means of multi-crossover among multiple parents in the good recruitment zone.

In summary, the crossover operators that many GPs have adopted typically involve the exchange of subtrees. Regardless of how the subtrees are swapped, the change in tree topology after the crossover is limited to the part of tree being exchanged in a single swap. Such incremental approach to tree evolution also tends to be slow. There is a need for the development of an unbiased and more uniform evolutionary algorithm for tree topologies

so as to produce a more drastic change of tree topologies during evolution by breaking up large tree. And this should improve the speed of convergence.

### **2.3 Other Application Specific Graph Evolutionary Algorithms**

There are also other EAs that are developed to evolve graphs and these EAs are tailored to suite specific problems such as arithmetic circuit design [54], design planning [7][8][9][10], chemical molecule design [18], and computer network optimization [17].. These algorithms are very different from each other and cannot be easily generalized. The last three application domains are studied in this thesis for their popularity of application. The state of the art encoding scheme for graphs and their evolution for these specific domains are discussed in Sections 5.1, 8.1 and 9.1 respectively. They are compared with EvoGraph encoding and evolution in the same chapter.

This thesis presents a general graph encoding scheme for graphs of different topologies of graphs including ANNs or trees, directed or undirected etc.. Based on this general encoding scheme, an EA, called EvoGraph, is devised to tackle various problems for which solutions can be represented as graphs. The details of EvoGraph are given in the next two chapters.

---

## EvoGraph Encoding Scheme for Different Graph Topologies

---

All graphs can be represented in their adjacency matrices. The adjacency matrix of a graph represents the graph nodes and their connectivity at the same time. For EvoGraph, a graph is encoded in its adjacency matrix. Though these matrices can also be encoded in linear-string chromosomes which simple GAs can operate on by concatenating the rows to form a linear array, it can be a rather computationally clumsy representation. Furthermore, if linear-string chromosomes are used, the connectivity between nodes cannot be read directly and a special decoding phase has to be added to convert linear-string chromosomes back into a graph.

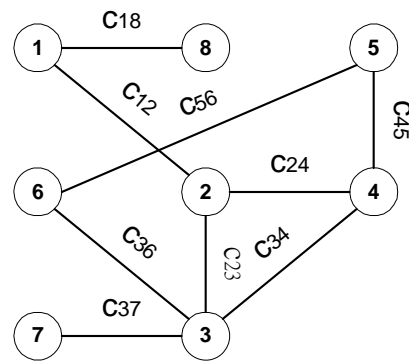
One advantage of the EvoGraph using an adjacency matrix encoding scheme is that an effective crossover operator that EvoGraph adopts can be more easily implemented with it. Also, another advantage of encoding a graph in adjacency matrix is that “repairing” a matrix after crossover to ensure connectivity can be much easier. In the following sections, the adjacency matrix representation scheme is described in details.

### 3.1 Encoding of Graphs in Adjacency Matrix

Given a graph represented as  $G(V, E)$  where  $V$  is the set of vertices and  $E$  is the set of edges in  $G$ . We can construct its adjacency matrix in such a way that if there exists a connection from  $v_i \in V$  to vertex  $v_j \in V$ ,  $i \neq j$  in the graph, then the value of the cell at the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column,  $c_{ij}$  is set to 1. Otherwise, if there is no connection between them, it

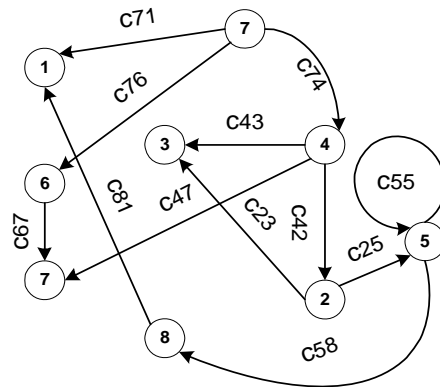
is set to 0. If the edges are labeled,  $c_{ij}$  can be assigned the edge label instead. Without loss of generality, if the graph is directed, then  $c_{ij}$  represents a directed edge from  $v_i \in V$  to vertex  $v_j \in V$ . Examples on the encoding of undirected and directed graph are shown below in Figures 1 and 2.

	1	2	3	4	5	6	7	8
1	-	$c_{12}$	0	0	0	0	0	$c_{18}$
2	-	-	$c_{23}$	$c_{24}$	0	0	0	0
3	-	-	-	$c_{34}$	0	$c_{36}$	$c_{37}$	0
4	-	-	-	-	$c_{45}$	0	0	0
5	-	-	-	-	-	$c_{56}$	0	0
6	-	-	-	-	-	-	0	0
7	-	-	-	-	-	-	-	0
8	-	-	-	-	-	-	-	-



**Figure 1:** Undirected graph encoding example

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	$c_{23}$	0	$c_{25}$	0	0	0
3	0	0	0	0	0	0	0	0
4	0	$c_{42}$	$c_{43}$	0	0	0	$c_{47}$	0
5	0	0	0	0	$c_{55}$	0	0	$c_{58}$
6	0	0	0	0	0	0	$c_{67}$	0
7	$c_{71}$	0	0	$c_{74}$	0	$c_{76}$	0	0
8	$c_{81}$	0	0	0	0	0	0	0



**Figure 2:** Directed graph encoding example



### 3.2 Encoding Trees

Given a number of nodes  $|V|$ , a tree is the minimal graph that has  $|E| = |V| - 1$  number of edges. The adjacency matrix of a tree with  $|V|$  nodes and  $|E|$  directed edges starting from the root and descending downwards has the properties as shown in Table 1. Note all trees exhibit such properties in their adjacency matrices but an adjacency matrix having such properties may not always produce a tree topology. An example on tree adjacency matrix encoding a tree and adjacency matrix having properties in Table 1 but not representing a tree are shown in Figure 3(a) and Figure 3(b) respectively..

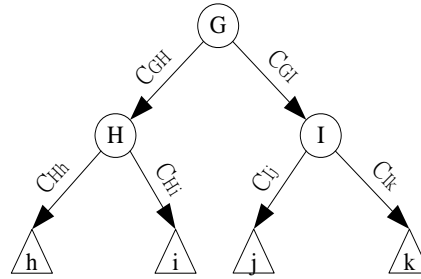
To check if an adjacency matrix represents a tree, one can follow the following procedure.

1. If an adjacency matrix does not possess properties in Table 1, it does not represent a tree topology.
2. Insert a cut-line between two columns at one time from the left to the right to separate the adjacency matrix into 2 submatrices. If it represents a tree, this will result in the loss of an edge. If not, the adjacency matrix does not represent a tree topology. For example, a cut between columns C and D of the adjacency matrix in Figure 3(b) does not result in the loss of an edge. Hence it does not represent a tree topology.

Properties in adjacency matrix	Topological properties in tree structure
1. For each node $i$ , there exist at least one $c_{ij} = 1$ or $c_{ki} = 1$ where $1 \leq j, k \leq  V $	Each node is connected at least by one arrow
2. There exist only one $c_{ij} = 1$ for all the columns in the matrix except the column of root node	There are $ V  - 1$ arrows in a tree and each non-root node has one number of arrow pointed from above
3. There exist one column with all $c_{ij} = 0$	The root node does not have arrow pointing toward it
4. There exist some rows with $c_{ij} = 0$	The terminal nodes have no arrows originating from them
7. If $c_{ij} = 1$ , then $c_{ji} \neq 1$ and vice versa	There is no reverse arrow pairs connecting two nodes
6. The diagonal $c_{ii} = 0$	There is no loop for each node

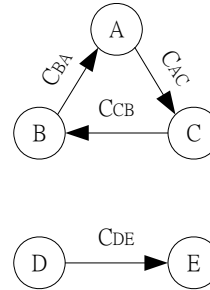
**Table 1:** Properties of a tree in adjacency matrix

	<b>G</b>	<b>H</b>	<b>I</b>	<b>h</b>	<b>i</b>	<b>j</b>	<b>k</b>
<b>G</b>	0	$c_{GH}$	$c_{GI}$	0	0	0	0
<b>H</b>	0	0	0	$c_{Hh}$	$c_{Hi}$	0	0
<b>I</b>	0	0	0	0	0	$c_{Ij}$	$c_{Ik}$
<b>h</b>	0	0	0	0	0	0	0
<b>i</b>	0	0	0	0	0	0	0
<b>j</b>	0	0	0	0	0	0	0
<b>k</b>	0	0	0	0	0	0	0



**Figure 3(a):** Tree encoding example

	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>
<b>A</b>	0	0	$c_{AC}$	0	0
<b>B</b>	$c_{BA}$	0	0	0	0
<b>C</b>	0	$c_{CB}$	0	0	0
<b>D</b>	0	0	0	0	$c_{DE}$
<b>E</b>	0	0	0	0	0



**Figure 3(b):** Example on adjacency matrix with properties in Table 1 but not representing tree topology

### 3.3 Encoding Artificial Neural Network

EvoGraph is not designed to conduct ANN operations with the feed forward and back propagation process. The intention is to evolve the optimal ANN topology for its operation. The input, output, and hidden nodes of ANN are encoded in adjacency matrix by assigning the rows and columns to present different layers. Only the above diagonal elements in the upper left portion of the matrix are used to indicate connections. This is because most of the ANNs are bipartite graphs and nodes in each layer of ANN will not connect with itself. The general form of adjacency matrix for ANN and an example of

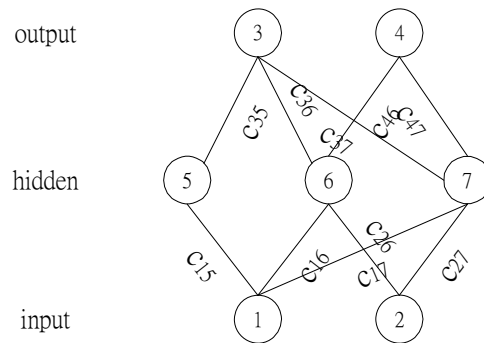
encoding is shown below.  $c_{ij}$  is the weight of an edge. The direction of edges is trivial for evolving topologies of ANN. Hence, undirected graph encoding similar to Figure 1 is used. For recurrent ANN, which is a special type of ANN, it has tree topology and can be encoded using the adjacency matrix in Section 3.2

	Input nodes	Output nodes	Hidden nodes
Input nodes	-	-	$c_{ij}$
Output nodes	-	-	$c_{ij}$
Hidden nodes	-	-	-

Figure 4: General form of adjacency matrix for encoding ANN

	1	2	3	4	5	6	7
1	-	-	-	-	$c_{15}$	$c_{16}$	$c_{17}$
2	-	-	-	-	0	$c_{26}$	$c_{27}$
3	-	-	-	-	$c_{35}$	$c_{36}$	$c_{37}$
4	-	-	-	-	0	$c_{46}$	$c_{47}$
5	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-
7	-	-	-	-	-	-	-

Figure 5: ANN encoding example



---

## Reproduction Operators for EvoGraph

---

Given the graph encoding scheme, the reproduction operators can be easily described. Before they are described in details, it should be noted that, in the crossover and mutation operation as adopted by EvoGraph, the issue of graph connectivity needs to be tackled. Imposing penalty in the fitness function on invalid solutions (disconnected graphs) generated by evolution operators may lead to high inefficiency of the EA process because a lot of invalid solutions may need many generations to eliminate [133]. We use an appropriate data structure (adjacency matrix) and specialized evolution operators to take care of the connectivity constraint.

In order to ensure that a graph remains connected after the reproduction operators are applied to it, EvoGraph use an unbiased repair algorithm by incorporating a randomly generated spanning tree (see the procedures in Section 4.2) to the intermediate degenerate graphs right after the crossover or mutation. This way, a disconnected graph can be connected. Given a number of nodes, the spanning tree is the smallest graph that can be embedded in all graphs. Embedding of a spanning tree in a graph being generated during the evolutionary process does not cause any bias in topology in a graph. For example, in the initialization process, given  $|V|$  and  $|E|$ , graph generation begins with the generation of spanning tree to connect all the  $|V|$  number of nodes with  $|V| - 1$  edges, and then add edges at random along the way.

## 4.1 Generation of Spanning Tree

As the EvoGraph operates on adjacency matrices, spanning tree generation is conducted on adjacency matrix. The generation of spanning tree from a set of unconnected nodes in an adjacency matrix is illustrated by an example below. For the purpose of illustration,  $c_{ij} = 1$  or  $0$  where  $c_{ij} = 1$  if a directed edge connection from node  $v_i$  to node  $v_j$  exists and  $c_{ij} = 0$  indicates no connection.

Before the procedure to embed a decision tree in an adjacency matrix is given, let us define the followings.

*Definition 1:* An *Origin node* ( $vo$ ) is defined as a node in a graph adjacency matrix with all '0's in the column. This means that there is no directed edge connection from another node to an *origin node* in a graph. An example of an *origin node* is the root node of a tree. It will be illustrated later that, during the process of crossover of trees, degenerate trees can be produced in the intermediate stages and there can be more than one *origin node* in a degenerate tree. However, the process of spanning-tree generation can also start off with a set of  $vos$  of a degenerate tree.

*Definition 2:* A *Descendant node* ( $vd$ ) is defined as a node in a graph adjacency matrix with a single '1' in one cell and '0' in all other cells in the column. This represents a directed edge connection from another node to a *descendant node*. Examples of *descendant node* include internal nodes and leaf nodes of a tree.

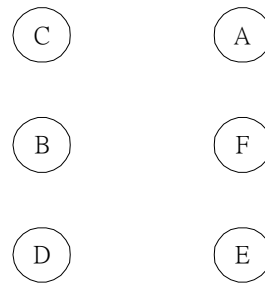
Given the above definitions and let  $V$  be a set of nodes for the spanning tree,  $VO \subset V$  be the set of *origin nodes* in  $V$ . If  $V$  is in the starting condition, where all the

nodes are unconnected,  $V=VO$ , otherwise,  $VO \subset V$  and  $VO \neq V$ . Let also  $N(X)$  represents number of elements in set  $X$ . The procedure for constructing an embedded spanning tree can therefore be given as follows:

1. Initialize an adjacency matrix with all column and row labels the same as the nodes in  $V$ . Set all cells of the adjacency matrix to '0'.
2. Select at random a node  $v_r$  from  $VO$ .
3. Exclude  $v_r$  from  $VO$  so that  $VO=VO \setminus v_r$ ;
4. Generate at random a natural number  $k$  where  $1 \leq k \leq N(VO)$ .
5. Select  $k$  number  $v$ os in  $VO$ . Create a set  $U$  to include all the selected  $v$ os so that  $U = \{v_1, v_2, \dots, v_i, \dots, v_k\}$ ;
6. for  $i = 1$  to  $k$ ,  $c_{v_r, v_i} = 1$  so that directed edges are now created to connect  $v_r$  to all nodes  $v_i$  in  $U$  in the adjacency matrix. All nodes in  $U$  are now  $v$ ds.
7. Revise  $VO$  to exclude all the  $v$ ds so that  $VO=VO \setminus U$ ;
8. Include  $v_r$  again in  $VO$  for the next round of *origin node* selection.  $VO = VO \cup v_r$ ;
9. Empty  $U$ .  $U = \{\}$ ;
10. Repeat step 2 to 9 until  $N(VO)=1$ . The ending condition where there is only one root node in the set of *origin nodes*.

An example of spanning tree generation from an adjacency matrix using the above procedure is shown below. Given a set of unconnected nodes  $V = \{A, B, C, D, E, F\}$ . as shown in Figure 6(a).

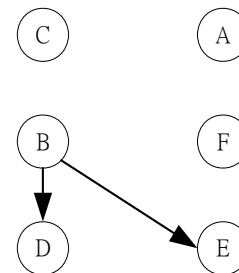
	B	E	D	A	F	C
B	0	0	0	0	0	0
E	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
F	0	0	0	0	0	0
C	0	0	0	0	0	0



**Figure 6(a):** Initialization of spanning tree adjacency matrix. All nodes have column cells '0'. Hence all nodes are *vos*

1. Let  $VO=V=\{A, B, C, D, E, F\}$
2. Select at random an *origin node*  $v_r='B'$  from  $VO$ .
3.  $VO=VO\setminus B$ ;  $VO = \{A, C, D, E, F\}$ .
4. Generate at random a natural number  $1 \leq k \leq N(VO) = 5$ , say  $k=2$ .
5. Select at random 2 *vos* from  $VO$  and include them in a set  $U$ , say,  $U = \{D, E\}$ .
6. Create directed edges  $c_{BD} = 1, c_{BE} = 1$ . Now ' $D$ ', ' $E$ ' are no longer *vos*. Each of them has a '1' in their columns in the adjacency matrix. They are *vds*.

	B	E	D	A	F	C
B	0	1	1	0	0	0
E	0	0	0	0	0	0
D	0	0	0	0	0	0
A	0	0	0	0	0	0
F	0	0	0	0	0	0
C	0	0	0	0	0	0



**Figure 6(b):** Node ' $B$ ' is selected to originate directed edges to *vos* ' $D$ ' and ' $E$ '

7. Revise  $VO$  to exclude *vds* in  $U = \{D, E\}$ .  $VO = VO \setminus U = \{A, C, F\}$ .

8. Include  $v_r='B'$  again in  $VO$  for the next round of *origin node* selection.

$$VO = VO \cup v_r = \{A, B, C, F\};$$

9. Empty  $U$ .  $U=\{\}$ ;

10.  $N(VO) = 4 \neq 1$ . Carry out next round of *origin node* selection.

11. Select at random an *origin node*  $v_r= 'A'$  from  $VO$ .

$$VO=VO \setminus A; VO = \{B, C, F\}.$$

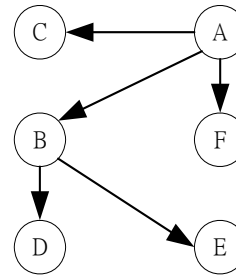
13. Generate at random a natural number  $1 \leq k \leq N(VO) = 3$ , say  $k=3$ .

14. Select at random 3 *vos* from  $VO$  and include them in a set  $U$ , say,  $U = \{B, C,$

$F\}$ . Create directed edges  $c_{AB} = 1, c_{AC} = 1, c_{AF} = 1$ . Now ' $B$ ', ' $C$ ', ' $F$ ' are no longer *vos*.

Each of them has a '1' in their columns in the adjacency matrix. They are now *vds*.

	B	E	D	A	F	C
B	0	1	1	0	0	0
E	0	0	0	0	0	0
D	0	0	0	0	0	0
A	1	0	0	0	1	1
F	0	0	0	0	0	0
C	0	0	0	0	0	0



**Figure 6(c):** Node ' $A$ ' is selected to originate directed edges to *vos* ' $C$ ', ' $B$ ' and ' $F$ '

15. Revise  $VO$  to exclude the *vds* in  $U = \{B, C, F\}$ .  $VO = VO \setminus U = \{\}$ .

16. Include  $v_r='A'$  again in  $VO$  for the next round of *origin node* selection.

$$VO = VO \cup v_r = \{A\};$$

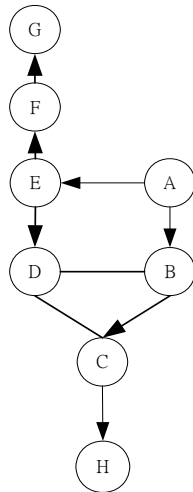
17. Empty  $U$ .  $U=\{\}$ ;

18.  $N(VO) = 1$ . Stop.



In complexity analysis, the worst case scenario of this spanning tree generation algorithm is to add one edge at a time until a spanning tree is formed. In this worst case scenario, a node is selected at random. Then a second node is selected out of  $|V| - 1$  nodes and a directed edge from the first node connects to it. The first node is an *on*. There are  $|V| - 2$  nodes left behind in the adjacency matrix to be scanned for *ons* for the next connection and so on. The order of time required for the spanning tree generation for this worst scenario is  $(|V| - 1) + (|V| - 2) + \dots + 1 = \frac{|V|(|V| - 1)}{2}$ . Hence, it can be deduced that the time for execution can be completed in not more than  $O(|V|^2)$ . The same time complexity applies to the repair processes in different crossovers described in Section 4.2 as they adopt the same algorithm for repair.

An example of embedded spanning tree in a graph is shown in Figure 7(a). The embedding of a spanning tree in the graph is shown in Figure 7(b) with corresponding edges shown as ‘1’ in the adjacency matrix. After creation of a spanning tree, more ‘1’s are then added to the cells of the adjacency matrix as additional edges are added to form a graph. They are presented as undirected edges in Figure 7(a) and ‘1’s in Figure 7(b). This process is used to generate the initial population of graphs for evolution. It also re-connects the exchanged degenerate subgraphs to generate offspring in the crossover process demonstrated in later sections.



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

(a) Graph with spanning tree embedment and node A as root

(b) Adjacency Matrix for directed graph with a '1's as the spanning tree and other edges added at random as '1'

**Figure 7:** An example of a graph adjacency matrix with spanning tree highlight

## 4.2 Crossover of Graphs

This operator involves the crossing of two parent graphs to produce a pair of children graphs in a random manner without imposed order and is the most general form of graph crossover used by EvoGraph. This is called *random crossover*.

### 4.2.1 Random Crossover of Graphs

This operator allows two parent graphs with different number of nodes to be crossed. A graph is invariant to the row and column permutation of its adjacency matrix. This permutation-invariant property of graphs does not exist in linear chromosomes in GA. In a linear chromosome, permutation of alleles will result in different properties and hence different fitness but this is not the case with graphs. The fitness of a graph is the same regardless of the permutation of its nodes even if the graph is represented as an adjacency matrix.

Given two parent graphs with different number of nodes to be crossed, they can be represented as  $G^{P1}(V^{P1}, E^{P1})$  and  $G^{P2}(V^{P2}, E^{P2})$  with corresponding adjacency matrices  $\mathbf{G}^{P1}$  and  $\mathbf{G}^{P2}$ , the operation can be described as follows. For illustration purpose, the value of elements in the adjacency matrix  $c_{ij} = 0$  or 1.

For each of  $G^{P1}$  and  $G^{P2}$ , the order of nodes in their corresponding adjacency matrices is randomly permuted prior to other operations. This is to ensure that there is no bias when nodes are selected for the reproduction operators. For each of  $\mathbf{G}^{P1}$  and  $\mathbf{G}^{P2}$ , a crossover point between two neighboring rows and columns in an adjacency matrix is selected randomly. One submatrix from each of the split parent matrix is then swapped to form two new child matrices. Connections within the submatrices are retained throughout the process while connections outside them are deleted. New edges are generated with repairing in each of the resulting matrices to form two children,  $\mathbf{G}^{C1}$  and  $\mathbf{G}^{C2}$ . The *random crossover* can be described step-by-step as follows.

1. Given two graphs,  $G^{P1}(V^{P1}, E^{P1})$  and  $G^{P2}(V^{P2}, E^{P2})$ , with vertex sets  $\{v_1^{P1}, v_2^{P1}, \dots, v_i^{P1}, v_{i+1}^{P1}, \dots, v_n^{P1}\}$  and  $\{v_1^{P2}, v_2^{P2}, \dots, v_j^{P2}, v_{j+1}^{P2}, \dots, v_m^{P2}\}$  respectively. For the two graphs, corresponding adjacency matrices  $\mathbf{G}^{P1}$  and  $\mathbf{G}^{P2}$  are constructed. The order of nodes is randomly permuted.
2. A crossover point in each of  $\mathbf{G}^{P1}$  and  $\mathbf{G}^{P2}$  is randomly selected.
3. Assume that the crossover point for  $\mathbf{G}^{P1}$  is between  $v_i^{P1}$  and  $v_{i+1}^{P1}$  and for  $\mathbf{G}^{P2}$  is between  $v_j^{P2}$  and  $v_{j+1}^{P2}$ , the lower right portions of these two adjacency matrices are then swapped so that the rows and columns corresponding to  $\{v_{i+1}^{P1}, \dots, v_n^{P1}\}$  are swapped with the rows and columns corresponding to  $\{v_{j+1}^{P2}, \dots, v_m^{P2}\}$  to form two matrices  $\mathbf{G}^{PC12}$  and

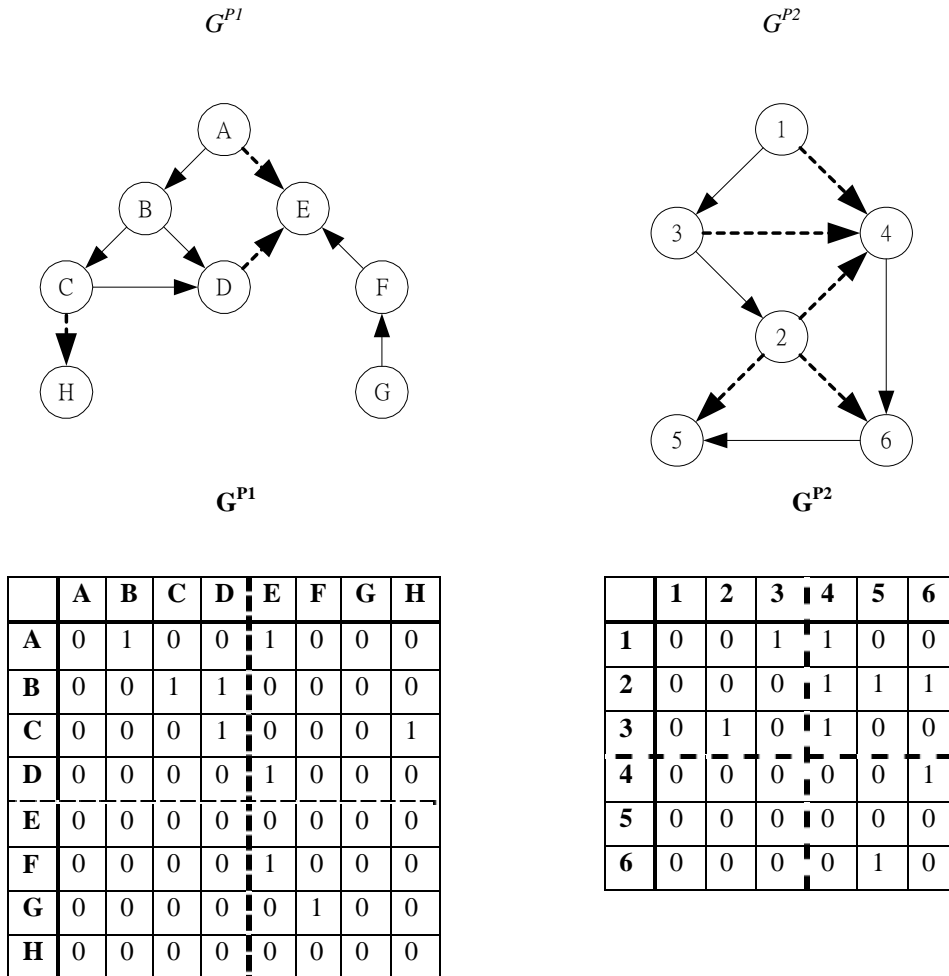
$\mathbf{G}^{\text{PC21}}$ . The valid vertex labels for  $\mathbf{G}^{\text{PC12}}$  are therefore given by  $\{v_1^{P1}, v_2^{P1}, \dots, v_i^{P1}, v_{j+1}^{P2}, \dots, v_m^{P2}\}$  and for  $\mathbf{G}^{\text{PC21}}$  by  $\{v_1^{P2}, v_2^{P2}, \dots, v_j^{P2}, v_{i+1}^{P1}, \dots, v_n^{P1}\}$ .

4. All cell entries in each of  $\mathbf{G}^{\text{PC12}}$  and  $\mathbf{G}^{\text{PC21}}$  are scanned to remove invalid edges in such a way that the edges connecting to the internal nodes within  $\mathbf{G}^{\text{PC12}}$  and  $\mathbf{G}^{\text{PC21}}$  are kept and those that connect an internal to an external nodes outside of  $\mathbf{G}^{\text{PC12}}$  and  $\mathbf{G}^{\text{PC21}}$  are removed.
5. The number of edges to be added to each of  $\mathbf{G}^{\text{PC12}}$  and  $\mathbf{G}^{\text{PC21}}$  respectively are then decided with a random number generator so that, for  $\mathbf{G}^{\text{PC12}}$ ,  $|V^{\text{PC12}}| - 1 \leq |E^{\text{PC12}}| \leq |V^{\text{PC12}}|C_2$  and for  $\mathbf{G}^{\text{PC21}}$ ,  $|V^{\text{PC21}}| - 1 \leq |E^{\text{PC21}}| \leq |V^{\text{PC21}}|C_2$ .
6. A spanning tree is generated at random in each matrix and the remaining edges to be added are generated randomly in such a way that they connect vertices from  $\{v_1^{P1}, v_2^{P1}, \dots, v_i^{P1}\}$  with those from  $\{v_{j+1}^{P2}, \dots, v_m^{P2}\}$  and from  $\{v_1^{P2}, v_2^{P2}, \dots, v_j^{P2}\}$  with those from  $\{v_{i+1}^{P1}, \dots, v_n^{P1}\}$ .
7. Once edge-generation is complete, two children graphs,  $\mathbf{G}^{\text{C1}}$  and  $\mathbf{G}^{\text{C1}}$ , are produced.

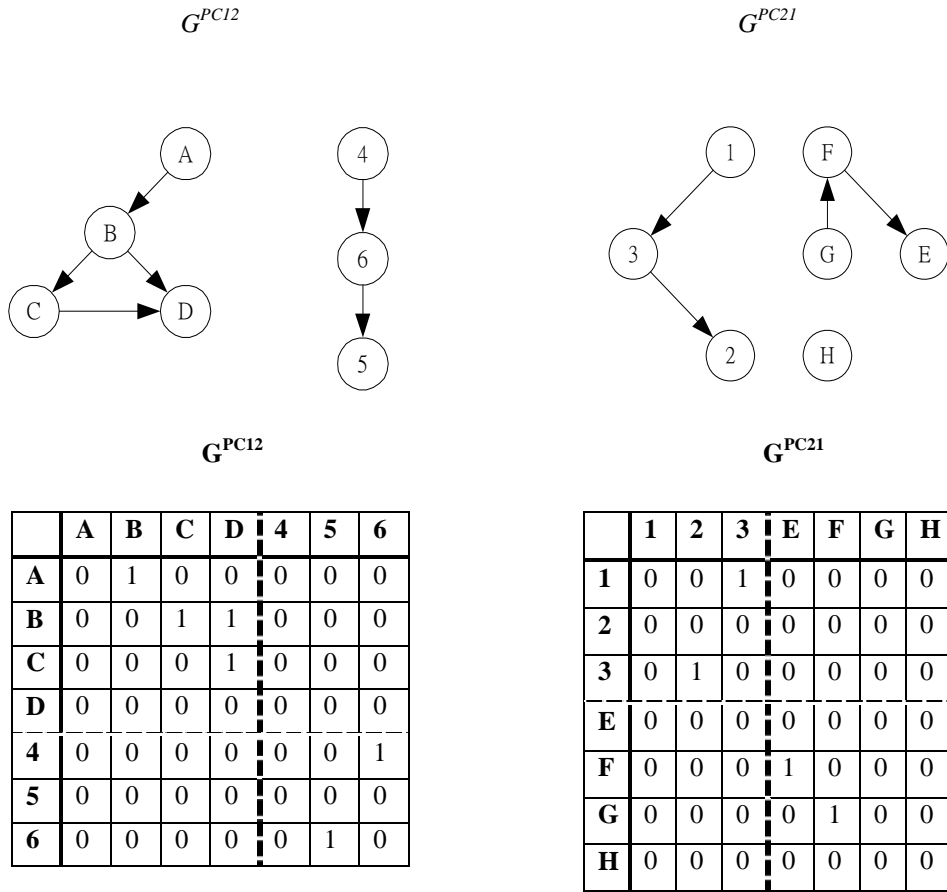
As an illustration of how the *random crossover* operates, let us consider an example in Figure 8 below.

1. Assume that the nodes of 2 graphs with adjacency matrices  $\mathbf{G}^{\text{P1}}$  and  $\mathbf{G}^{\text{P2}}$  respectively are randomly permuted prior to crossover.
2. A crossover point in each of  $\mathbf{G}^{\text{P1}}$  and  $\mathbf{G}^{\text{P2}}$  is then randomly selected.
3. The lower right portions of these two adjacency matrices are then swapped to form  $\mathbf{G}^{\text{PC12}}$  and  $\mathbf{G}^{\text{PC21}}$ .

4. All cell entries in each of  $G^{PC12}$  and  $G^{PC21}$  are scanned to remove invalid edges. Both of them are degenerate graph.



**Figure 8(a):** Nodes are randomly permuted in the adjacency matrices. A crossover point is randomly chosen for  $G^{P1}$  and  $G^{P2}$ . Invalid edges to be removed are shown in dotted lines. Corresponding graphs are shown in the above adjacency matrices



**Figure 8(b):** Invalid edges are removed after swapping of lower right portions of  $G^{P1}$  and  $G^{P2}$ . Degenerate graphs  $G^{PC12}$  and  $G^{PC21}$  are formed

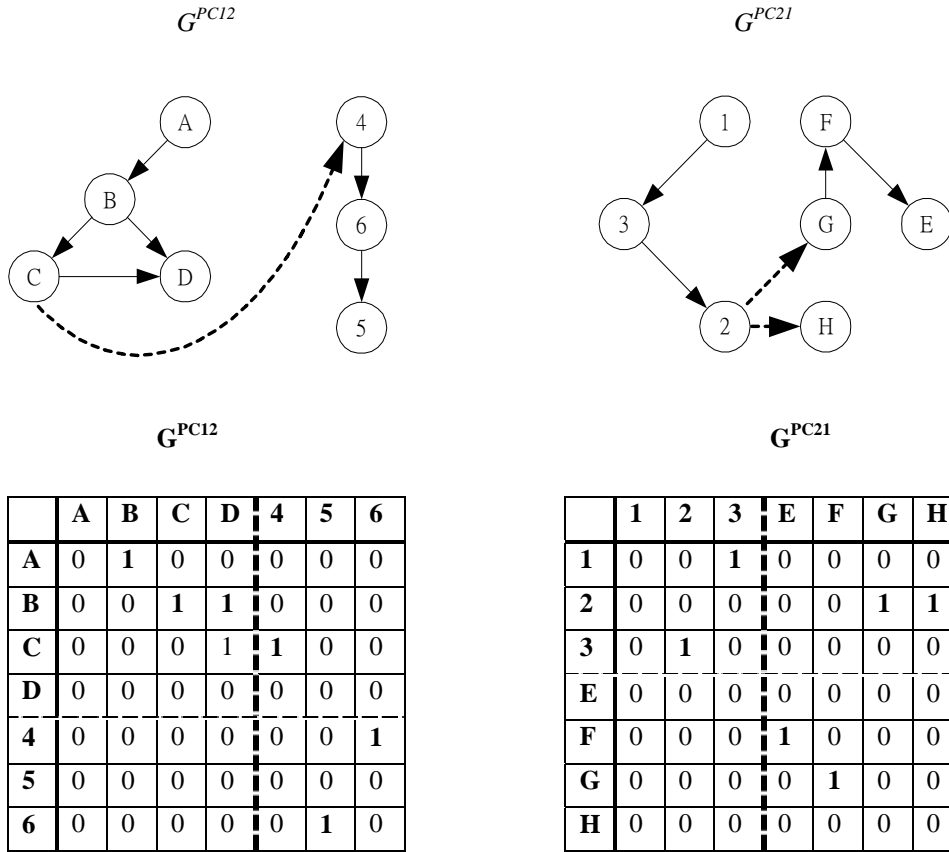
5. The number of edges to be added to each of  $G^{PC12}$  and  $G^{PC21}$  respectively are then decided with a random number generator so that,

$$\text{for } G^{PC12}, |V^{PC12}| - 1 = 6 \leq |E^{PC12}| \leq |V^{PC12}| C_2 = 21 \text{ and}$$

$$\text{for } G^{PC21}, |V^{PC21}| - 1 = 6 \leq |E^{PC21}| \leq |V^{PC21}| C_2 = 21.$$

In this example, say,  $|E^{PC12}| = 10$ ,  $|E^{PC21}| = 6$ .

6. A spanning tree is generated at random for the nodes in each matrix according to the process in Section 4.1 and superimpose on the adjacency matrix of each of the degenerate graphs to re-connect them. A spanning tree is now embedded in each of the graph.



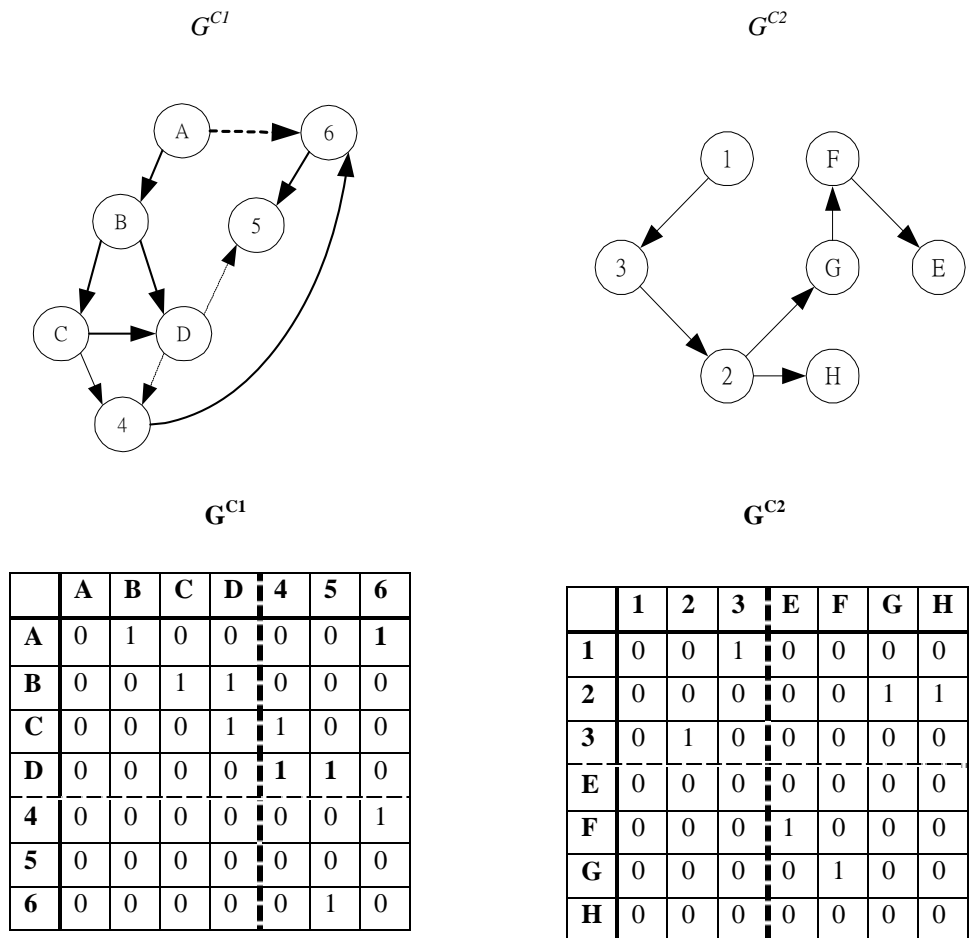
**Figure 8(c):** Re-connect degenerate graphs by superimposing a spanning tree generated at random as in section 4.1. Spanning tree edges are shown as ‘1’s in the adjacency matrices. New edges added to re-connect the degenerate graphs are shown as dotted line in the graph

$G^{PC12}$  has 7 edges. The requirement of  $|E^{PC12}| = 10$  in step 5 is not satisfied in  $G^{PC12}$ .

New edges should be added to  $G^{PC12}$  to fulfill the requirement. Note that if the edges in the resulting graphs are more than specified, edges that are not in the embedded spanning tree in each graph can be deleted at random to achieve the target.

7. Three new edges are randomly added to  $G^{PC12}$ , say,  $c_{D4} = 1, c_{D5} = 1, c_{A6} = 1$  to form  $G^{C1}$ .

$G^{C2}$  is formed without addition of edges.



**Figure 8(d):** New edges  $c_{D4}=1, c_{D5}=1, c_{A6}=1$  in '1' are added randomly to  $G^{PC12}$ . Children graphs  $G^{C1}$  and  $G^{C2}$  are formed.

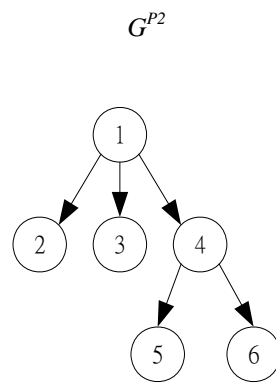
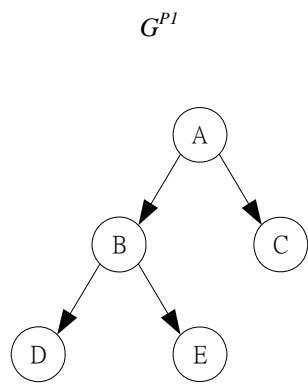
#### 4.2.2 Random Crossover of Trees

To illustrate how the *random crossover* works with trees, let us consider an example below. For this example, a directed graph is used for illustration.

1. Select two parent trees  $G^{P1}$  and  $G^{P2}$  with adjacency matrices  $G^{P1}$  and  $G^{P2}$  for crossover.

Permute the order of nodes in the adjacency matrix at random.





$G^{P1}$

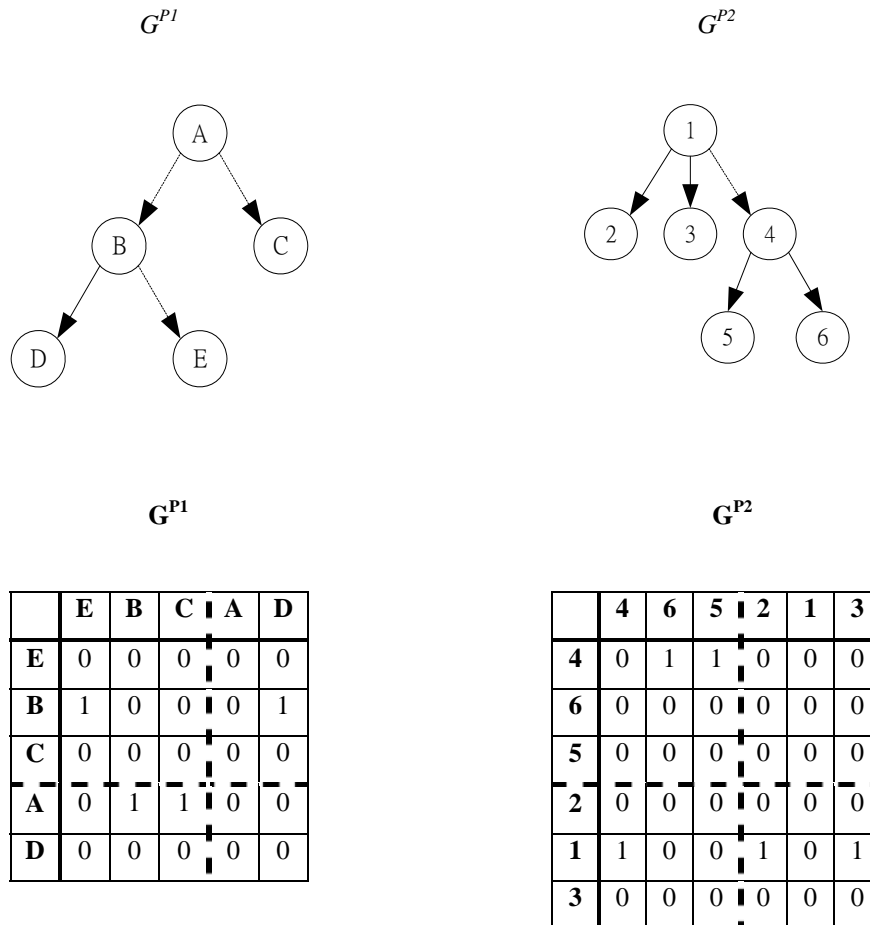
$G^{P2}$

	<b>E</b>	<b>B</b>	<b>C</b>	<b>A</b>	<b>D</b>
<b>E</b>	0	0	0	0	0
<b>B</b>	1	0	0	0	1
<b>C</b>	0	0	0	0	0
<b>A</b>	0	1	1	0	0
<b>D</b>	0	0	0	0	0

	<b>4</b>	<b>6</b>	<b>5</b>	<b>2</b>	<b>1</b>	<b>3</b>
<b>4</b>	0	1	1	0	0	0
<b>6</b>	0	0	0	0	0	0
<b>5</b>	0	0	0	0	0	0
<b>2</b>	0	0	0	0	0	0
<b>1</b>	1	0	0	1	0	1
<b>3</b>	0	0	0	0	0	0

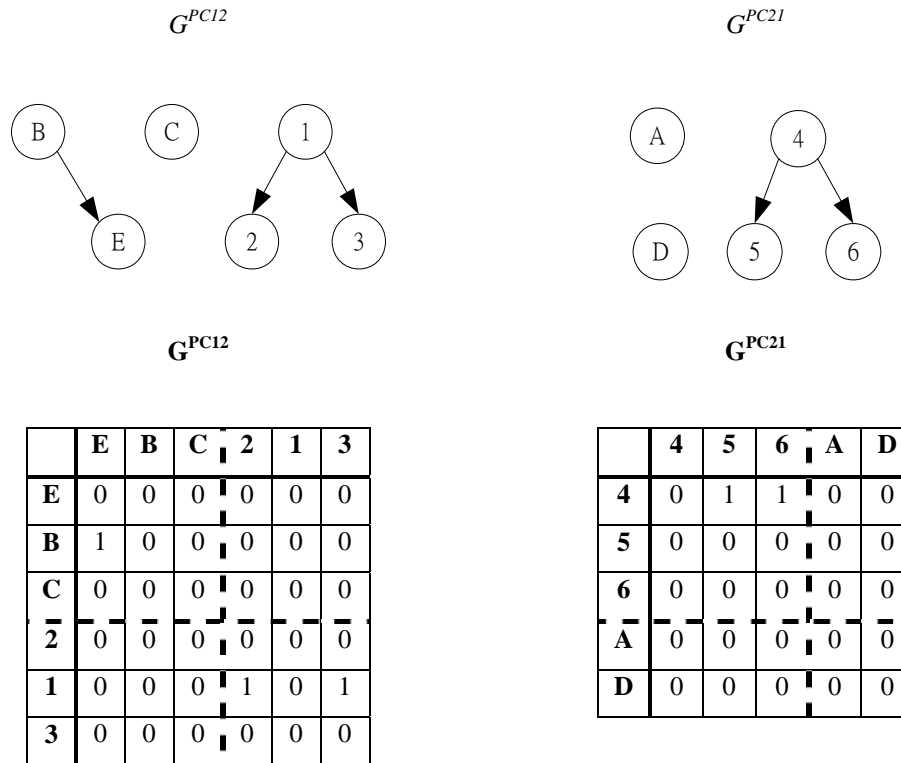
**Figure 9(a):** Encoding trees in adjacency matrices with order of nodes permuted at random

2. Select a crossover point in the matrices at random. The nodes and terminals are divided into two sub-groups in each parent accordingly.



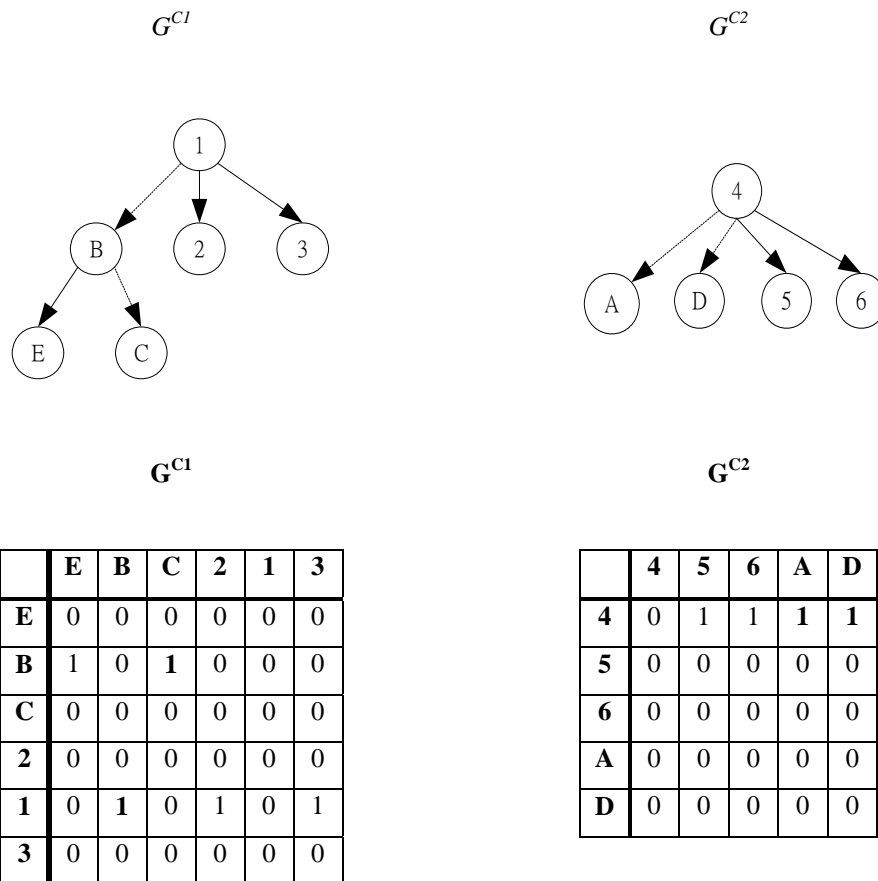
**Figure 9(b):** Randomly choose crossover points and sub-groups in the adjacency matrices. The edges to be removed are shown in dotted lines.

3. Remove invalid edges after exchange sub-groups to form  $G^{PC12}$  and  $G^{PC21}$ .



**Figure 9(c):** Exchange sub-groups between two adjacency matrices and remove invalid edges. Two degenerate trees are formed.

4. Generate spanning tree to re-connect all the nodes using the steps as described in Section 4.1 to form children trees,  $G^{C1}$  and  $G^{C2}$ . In this case, the spanning tree is generated from a degenerate tree with some origin nodes,  $vos$ , with all '0' in the column of the adjacency matrices  $G^{PC12}$  and  $G^{PC21}$  in Figure 9(c). This means that there is no ancestor node from the tree layer above connected to this node (Nodes '1', 'B', 'C' in  $G^{PC12}$  and nodes '4', 'A', 'D' in  $G^{PC21}$ ). Spanning tree is generated at random to re-connect each degenerated tree according to the process in Section 4.1 to form children trees  $G^{C1}$  and  $G^{C2}$ .



**Figure 9(d):** Generate spanning tree according to the process in section 4.1 to re-connect all nodes. New edges are shown as ‘1’s in adjacency matrices and dotted directed edges in trees  $G^{C1}$  and  $G^{C2}$

This is the result of one *random crossover* of trees in EvoGraph. The children trees are substantially different from their parents. It is apparent that one GP crossover involving subtree exchange between parents cannot achieve the result of one *random crossover* of EvoGraph. As shown from an example in the next section, to obtain the results using traditional swapping-subtree crossover in GP, more than one GP operation plus appropriate combination of mutations are required. The *random crossover* has the following properties when used with trees.

1. It is unbiased in the sense that each node and edge has equal chance of being swapped.

There is more random regrouping of nodes and edges with *random crossover* operator. The GP crossovers with subtree exchanges may need more steps to attain the same result of just one crossover with this algorithm. This will be illustrated by an example in Section 4.2.3.

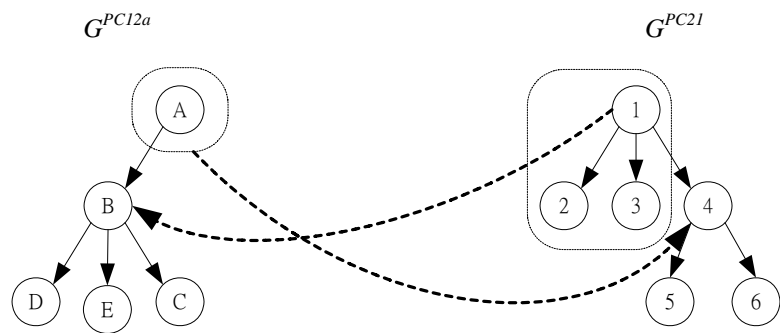
2. The selection of a node as crossover point does not imply the exchange of subtree below it. Hence, redundant codes in the subtree below the crossover point may not be carried over to the next generation. This may help to alleviate the problem on bloat as in the case with GP.
3. Several researchers have shown that the basic crossover operator in GP is inclined to converge in the nodes near the tree root [51][52][53]. It is difficult for crossover in GP with subtree exchange to change the “incorrect” node in the root structure without also changing all the structure below that node. EvoGraph is not constrained in this respect. The root of a parent can be easily separated from the nodes below it in a single crossover operation.

### **4.2.3 Comparison between Standard GP and Random Crossover of Trees in EvoGraph**

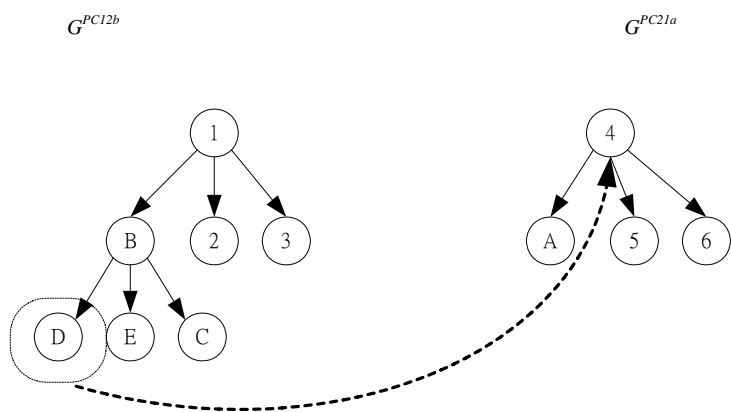
In order to illustrate how EvoGraph can be used to solve the kind of problems typically dealt with using GPs, let us consider the following example where GP with subtree exchange is applied to the same parents in 4.2.2. The example describes one of the possible paths step by step to arrive at the same result in 4.2.2. It is demonstrated that more than one crossover and mutation are required to achieve the same result.



**Figure 10(a):** Start condition of GP crossover. Node 'C' of  $G^{P1}$  is mutated to connect to node 'E'



**Figure 10(b):** First GP crossover with swapping of subtree '1-2-3' and node 'A'



**Figure 10(c):** Second GP crossover with swapping of node 'D'



**Figure 10(d):** Children trees same as Figure 9(d)

#### 4.2.4 Random Crossover of Artificial Neural Networks

To use EvoGraph with ANN, let us consider encoding an ANN as a graph with three sets of nodes, the input nodes, the output nodes and the hidden nodes. There is no edge connection between nodes within each set. This principle is used to generate ANNs in the process of crossover. During the crossover, the number of input nodes and output nodes are fixed. The hidden nodes and their edges are swapped at random. Using the notations in 4.2.1, the random crossover of ANN is illustrated by an example below. For illustration purpose, the weight of all edges is assumed to be 1.

Let  $\{i_1, i_2, \dots, i_p\}$  be the set of input layer nodes

$\{h_1, h_2, \dots, h_k\}$  be the set of hidden layer nodes

$\{o_1, o_2, \dots, o_q\}$  be the set of output layer nodes

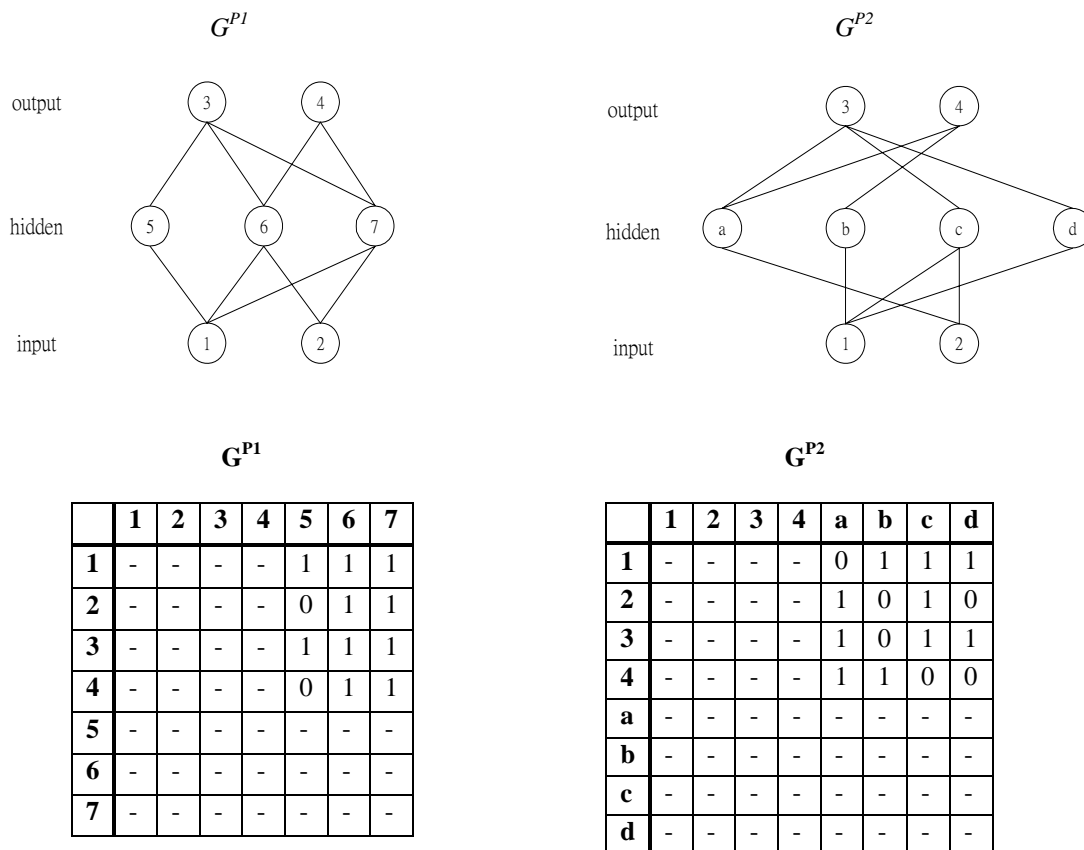
1. Concatenate  $\{i_1, i_2, \dots, i_p\}$ ,  $\{h_1, h_2, \dots, h_k\}$ ,  $\{o_1, o_2, \dots, o_q\}$  to form the rows and column labels of the adjacency matrix  $\{i_1, \dots, i_p, h_1, \dots, h_k, o_1, \dots, o_q\}$  arranged in the same order. Set all cells in the adjacency matrix to be '0'.

2. for  $j = 1 : k$ ,
  - generate natural numbers  $m, n$  at random where  $1 \leq m \leq p, 1 \leq n \leq q$ ;
  - insert  $m$  numbers '1's at random to cells in column  $h_j$ , row  $i_1$  to  $i_p$ ;
  - insert  $n$  numbers '1's at random to cells in column  $h_j$ , row  $o_1$  to  $o_q$ ;
- end

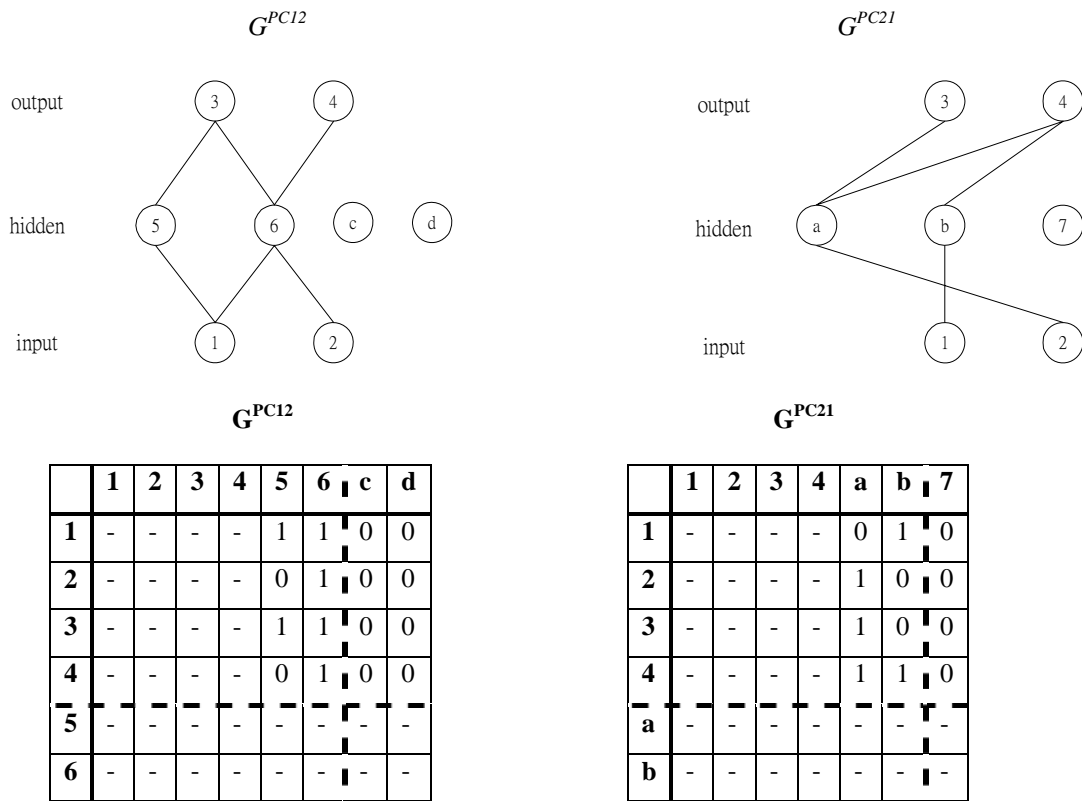
Step 1 is to initiate an adjacency matrix for ANN. Connections are not made at this step and all entries in the adjacency are set to zeros. Step 2 connects the hidden layer nodes with the input layer nodes and the output layer nodes. This step will also be used for reconnection of degenerate ANN after the exchange of subgraphs in the *random crossover* process of ANN illustrated below. During the crossover, the number of input nodes and output nodes are fixed. The hidden nodes and their edges are swapped at random. The *random crossover* of neural network is illustrated by an example below.

1. Select two parent ANNs  $G^{P1}, G^{P2}$  with adjacency matrices  $\mathbf{G}^{P1}$  and  $\mathbf{G}^{P2}$  (Figure 11(a)).
2. Select crossover point at random at the hidden layer in  $\mathbf{G}^{P1}$  and  $\mathbf{G}^{P2}$ . Split  $\mathbf{G}^{P1}$  and  $\mathbf{G}^{P2}$  into submatrices at the crossover point, exchange the submatrices and delete invalid edges to form  $\mathbf{G}^{PC12}$  and  $\mathbf{G}^{PC21}$ .





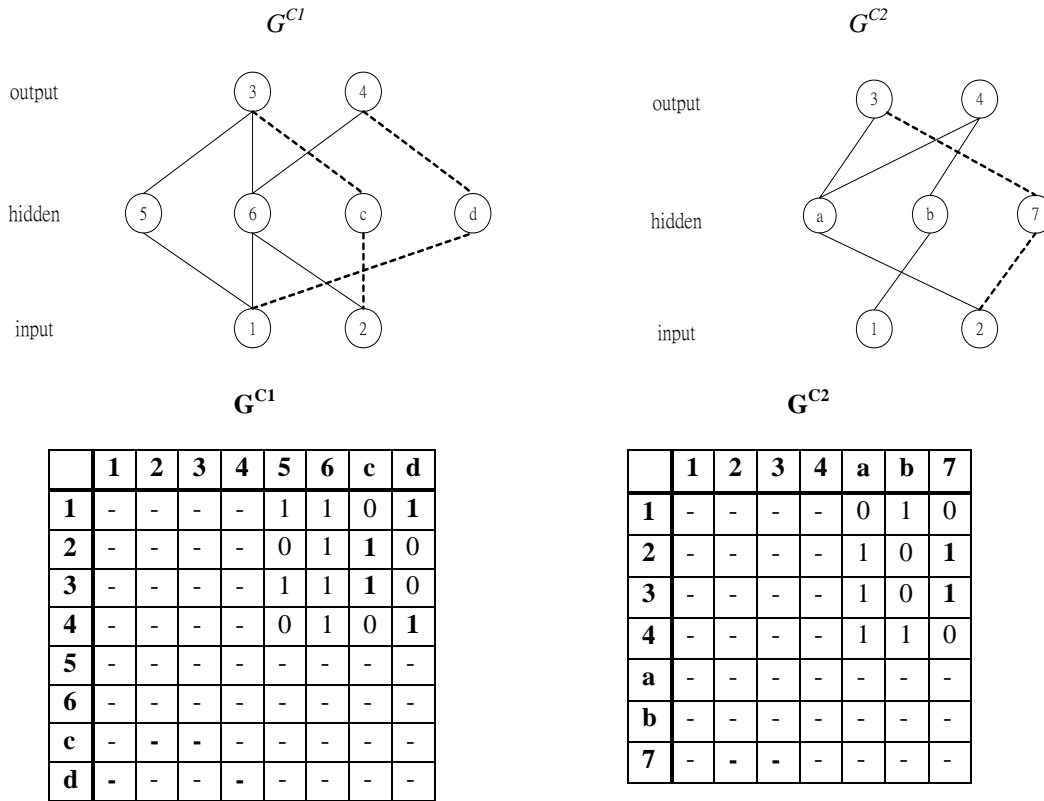
**Figure 11(a):** Encoding parent ANNs in adjacency matrices



**Figure 11(b):** Select crossover point at random, exchange submatrices and delete invalid edges to form  $G^{PC12}$  and  $G^{PC21}$ . The corresponding ANN topologies are  $G^{PC12}$  and  $G^{PC21}$

- Re-connect at random the input- and output-layer nodes to the swapped hidden layer nodes by adding '1's at random to input node cells and output node cells in column 'c' and 'd' in  $G^{PC12}$  and column '7' in  $G^{PC21}$  to form children ANN,  $G^{C1}$  and  $G^{C2}$ . According to the encoding rule in Section 3.3, there should be at least one '1' in both the group of input node cells and output node cells for each hidden node column in the adjacency matrix. In this example,  $c_{2c}=1$  (input node connection) and  $c_{3c}=1$  (output node connection) for column 'c' (hidden node) in  $G^{C1}$ .  $c_{1d}=1$  (input node connection) and  $c_{4d}=1$  (output node connection) for column 'd' (hidden node) in  $G^{C1}$ .

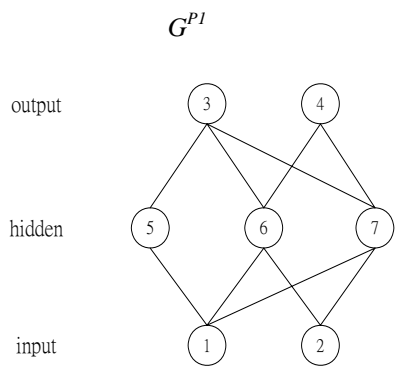
$c_{27}=1$  (input node connection) and  $c_{37}=1$  (output node connection) for column '7' (hidden node) in  $G^{C2}$ .



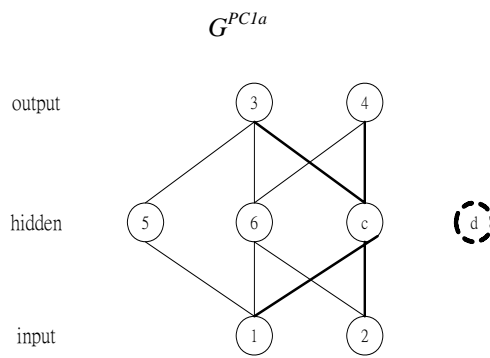
**Figure 11(c):** Re-generate edges to form children ANNs. The new edges are shown as '1's in  $G^{C1}$  and  $G^{C2}$  and dotted lines in  $G^{C1}$  and  $G^{C2}$

#### 4.2.5 Comparison between EP and EvoGraph on the Evolution of ANN Topology

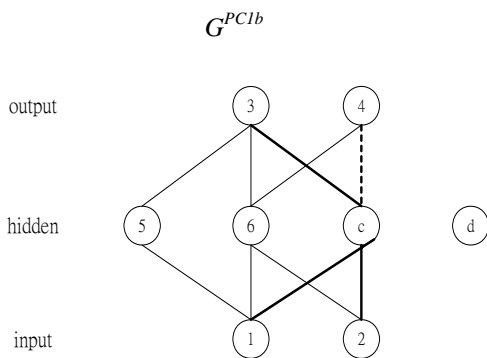
The use of EP to evolve ANN topology from  $G^{P1}$  in Figure 11(a) to  $G^{C1}$  in Figure 11(c) is illustrated as follow. EP only adopts mutation in its evolutionary process. It requires several mutation operations to achieve the result of one crossover operation in EvoGraph.



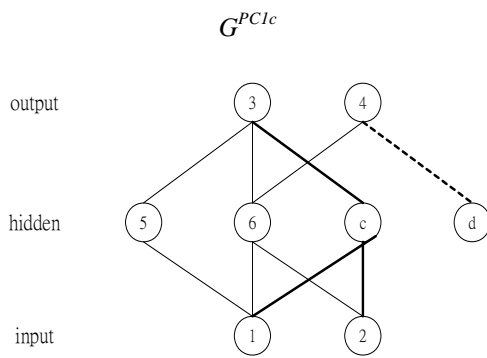
(a) Initial ANN  $G^{PI}$  before EP operation



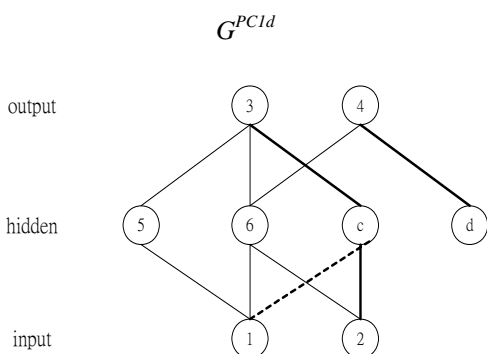
(b) Addition of one hidden layer node 'd'. Node '7' in hidden layer is changed to 'c' to achieve consistency with presentation in Figure 11.



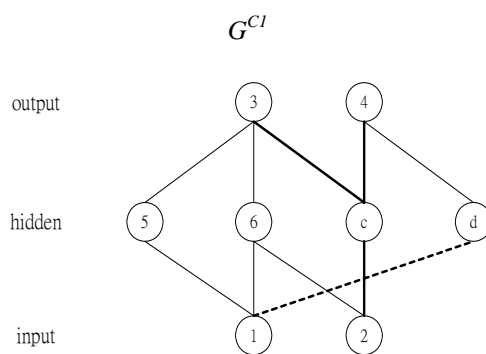
(c) Delete edge '4-c' from hidden layer to output layer



(d) Add edge '4-d' from hidden layer to output layer



(e) Delete edge '1-c' from input layer to hidden layer



(f) Add edge '1-d' from input layer to hidden layer

**Figure 12:** Steps on evolving ANN  $G^{PI}$  to  $G^{CI}$  using EP

*Random crossover* of ANN in EvoGraph has the following properties that are different from the existing evolutionary algorithms on ANN.

1. It produces a more vigorous change to the parent ANN topology in a single operation than its counterpart in EP.
2. Unlike the encoding of ANN into a linear chromosome by concatenating the rows of the adjacency matrix for the application of GA, which requires the length of the chromosome to be the square of the dimension of the adjacency matrix, EvoGraph has no requirement on the dimensions of the adjacency matrix.
3. The connectivity of the ANN can be read directly from the adjacency matrix. This is difficult to achieve with a simple linear chromosome without special encoding. Connectivity can be maintained by direct insertion of '1's in the adjacency matrix without special repair mechanism.
4. Unlike other indirect encoding that evolves instructions or rules for the generation of ANNs, EvoGraph adopts a direct encoding method. There is no requirement for additional mapping of the genotype, which is the subject of evolution in indirect encoding method, and the phenotype, the actual ANN itself.

### 4.3 Mutation of Graphs

Two basic mutation operators of graphs that cause changes in graph topologies are introduced. They are the *Number-of-edge* and *Number-of-Node* mutation operators. Similar to a GA, the intention of mutation is to bring about a breakthrough when the population converges to a suboptimal fitness. Based on this principle, other kinds of mutation can be further developed for different applications in later chapters.

#### 4.3.1 Number-of-Edge Mutation

The *Number-of-Edge* mutation operator allows us to increase or decrease the number of edges in a graph. It works by selecting an edge at random for deletion or addition. Its details are given below.

1. For a graph  $G^P(V^P, E^P)$  with edge set,  $E^P$ , we construct its corresponding adjacency matrices as  $\mathbf{G}^P$ .
2. Generate at random either '0' or '1'.
3. If '0' is generated, an edge in the graph  $G^P$  is selected and this is done by choosing a 'non-zero' cell in  $\mathbf{G}^P$  randomly. '1' is then deducted from the value of the selected cell to form a child graph  $G^C$  and its corresponding adjacency matrix  $\mathbf{G}^C$ . Check if  $\mathbf{G}^C$  is connected. If yes, stop. If not, the graph is broken down into two connected subgraphs and each has its own embedded spanning tree. Superimpose a spanning tree across the disjoint subgraphs to re-connect them.
4. If '1' is generated, an edge is added to the graph by adding a '1' to the value of a cell at random in  $\mathbf{G}^P$  to form  $G^C$  and  $\mathbf{G}^C$ .

An example of the *Number-of-Edge* mutation operator is given in Figure 13 below.

$G^P$ 

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

(a) For '0' case, select at random edge to be deleted

 $G^C$ 

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

(b) Delete selected edge. Check if PC is connected. If yes, stop. If no, superimpose a spanning tree at random

 $G^P$ 

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

(c) For '1' case, select at random cell to add an edge

 $G^C$ 

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

(d) add an edge to the selected cell

**Figure 13:** The Number-of-Edge Mutation operator illustrated.

### 4.3.2 Number-of-Node Mutation

The *Number-of-Node* mutation operator allows us to increase or decrease the number of nodes in a graph by one. It works by selecting a node at random for deletion or addition. Its details are given below.

1. For a graph  $G^P(V^P, E^P)$  with node set  $\{v_1^P, v_2^P, \dots, v_i^P, v_{i+1}^P, \dots, v_n^P\}$  we construct its corresponding adjacency matrices,  $\mathbf{G}^P$ .
2. Generate at random either '0' or '1'.
3. If '0' is generated, a node,  $v_i^P$ , in  $\mathbf{G}^P$  is selected at random.
4.  $v_i^P$  and all edges connected to  $v_i^P$  are deleted from  $G^P$  to form  $G^C$  with corresponding adjacent matrix  $\mathbf{G}^C$ .
5. Connect nodes previously connected to  $v_i^P$  at random and the resulting node set of  $G^C$  is represented as  $\{v_1^P, v_2^P, \dots, v_{i-1}^P, v_{i+1}^P, \dots, v_n^P\}$ .
6. If '1' is generated, add node  $v_{n+1}^P$  to  $G^P$  and  $\mathbf{G}^P$  to form  $G^C$  and  $\mathbf{G}^C$  respectively with new node set represented as  $\{v_1^P, v_2^P, \dots, v_i^P, v_{i+1}^P, \dots, v_n^P, v_{n+1}^P\}$ . Edges are generated at random to connect  $v_{n+1}^P$  to other nodes in  $\mathbf{G}^C$ .

An example to illustrate the *Number-of-Node* mutation operator is given in Figure 14 below.



$$G^P$$

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

$$G^C$$

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

(a) Select the node B.

(b) For '0' case, delete the selected node B and connect node A and D which are previously connected to B

$$G^C$$

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

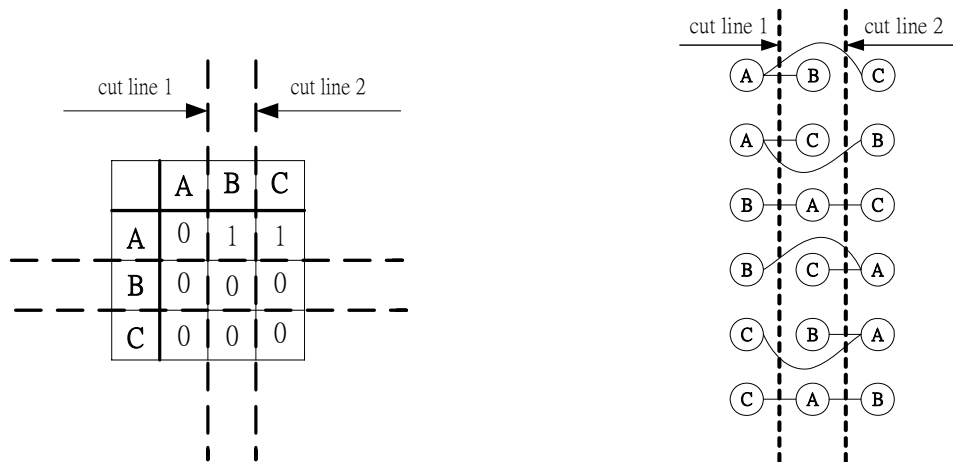
(c) For '1' case, add node I to  $G^P$  and edges connecting to nodes F and H

**Figure 14:** The Number-of-Node Operator illustrated.

#### 4.4 Mathematical Foundation for EvoGraph

A graph is invariant to the rows and columns permutation of its adjacency matrix. A cut on the adjacency matrix is equivalent to a cut in one of the many incidences of a graph which nodes aligned with the same node permutation as the adjacency matrix. Consider a simple example of a graph with 3 nodes A, B, and C. There are  $3! = 6$

permutations on the order of nodes. According to EvoGraph crossover, there are  $3-1 = 2$  positions for the cut line for crossover. The adjacency matrix with potential cut line positions and all the permutation of nodes in rows are shown in Figure 15.



(a) an adjacency matrix with an incidence of node permutation ABC and 2 potential cut lines

(b) all possible row permutations of nodes with 2 potential cut lines

**Figure 15:** Permutation invariant of adjacency matrix illustrated

Consider a node, say node A, is at the left most position of the row permutation of nodes. There are  $(3-1)! = 2!$  permutations for the remaining 2 nodes to take up the positions in the right of node A. They are in the order of B, C or C, B as illustrated in the first 2 rows in Figure 15(b). If node A is in the middle position, there are still 2! permutations for nodes B and C in the left and right of node A. This is illustrated in the third and fifth row in Figure 15(b). Similar logic applies when node A is in the right most position. The overall result is that there are exactly  $(3-1)! = 2$  nodes being node A in each column in Figure 15(b). This applies to node B and C.

In EvoGraph, the nodes in left hand side of the cut line will be retained and those in the right hand side will be swapped. Each cut line is selected at random with 3-1 possible

cuts. Hence the probability is  $\frac{1}{3-1} = 0.5$ . If cut line 1 is selected, there are  $(3-1)! = 2$  cases out of  $3! = 6$  cases that node A lies in the left of cut line 1 and being retained. At the same time, there are  $2 \times (3-1)! = 4$  cases out of  $3! = 6$  cases that node A will be swapped. That is, the probability of node A being swapped for cut line 1 is  $0.5 \times \frac{4}{6}$ .

When cut line 2 is selected with probability 0.5, there are  $2 \times (3-1)! = 4$  cases out of  $3! = 6$  cases that node A lies on the left of cut line 2 and be retained. At the same time, there are  $(3-1)! = 2$  cases out of  $3! = 6$  cases that node A will be swapped. That is, the probability of node A being swapped for cut line 2 is  $0.5 \times \frac{2}{6}$ .

Therefore, the total probability of node A being swapped in EvoGraph is

$$0.5 \times \left( \frac{4}{6} + \frac{2}{6} \right) = 0.5$$

This simple case can be generalized. Let  $|N|$  be the number of node of a graph  $G$  and node A as any node in  $G$ . The probability of a particular cut line being selected is  $\frac{1}{|N|-1}$ .

Counting from the left at the  $r - 1$ th cut line where  $2 \leq r \leq |N|$ , there is a probability

$$(|N| - r + 1) \times \frac{(|N| - 1)!}{|N|!}$$

that node A is at the right of the cutline and be swapped. The

total probability of any node A in  $G$  being swapped for all possible cut line locations is

$$\frac{1}{|N|-1} \sum_{r=2}^{|N|} (|N| - r + 1) \frac{(|N| - 1)!}{|N|!} = \frac{1}{|N|-1} \left( \frac{|N| (|N| - 1)}{2} \right) \frac{1}{|N|} = 0.5$$

Hence, the probability of any node in  $G$  being swapped is 0.5 for all possible cut line locations.

#### **4.4.1 Exploration Verses Exploitation**

Exploration and exploitation have always been an important consideration when choosing operators for evolutionary algorithms. In GA, there is an implicit thinking that high disruption caused by crossover destroys useful building blocks before the search space is adequately exploited and it should be avoided.

However, De Jong [67] pointed out that the maintenance of exploitation is at the expense of exploration, he demonstrated by example that exploration is more important when the population size is too small to provide the sampling accuracy for evolution [68]. Another advantage of uniform crossover is its defining length unbiasedness. It is equally disruptive to all defining lengths. This is supported by the mathematical proof in [63] that uniform crossover depends only on the order of alleles to be retained,  $o(s)$ , and it is independent to the defining length,  $\delta(s)$ , of a chromosome.

Furthermore, [67] demonstrated that the degree of disruption can be controlled by choosing the probability of an allele being swapped in uniform crossover,  $P_0$ . It was also shown that the most efficient  $P_0$  was 0.5. That is, every allele has an equal chance to stay or being swapped.

On the evolution of graphs, there has always been a problem of the need of exploring a large search space with a small population. There is a tremendous increase in the number of graph topologies with respect to the number of nodes. As shown in [69], the number of connected and graph topologies with respect to number of nodes is indicated

in Figure16. If the graphs are labeled, the increase in number of different graphs is more tremendous due to node permutation. Though the search space is huge, the size of the population in evolution cannot be large because this is detrimental to the efficiency of evolution. Hence, uniform crossover is more suitable for evolution of graphs. From the experience in GA, the probability of a node being swapped,  $P_o$ , is designed to be 0.5.

Number of nodes	1	2	3	4	5	6	7	8	9	10
Number of connected graph topologies	1	1	2	6	21	112	853	11,117	261,080	11,716,571

**Figure 16:** Number of connected graph topologies in w.r.t. number of nodes

Some work has been done on uniform crossover of trees, a special subset of graphs. Poli contributed a lot of literatures on the evolution of trees by using GP [53][49][70]. However, the schema theory for GP proposed [49] is based on the conventional hierarchical tree layer topology. The uniform crossover of trees in the Poli's literature requires matching of building block subtree topologies between 2 parents before crossover [48]. This involves heavy computation resource and hence affects the efficiency of evolution. Furthermore, the schema theory for GP proposed follows schema theory in GA but it is applied on the more complicated tree topology instead of a linear chromosome. The mathematical expressions involve heavy calculations on probabilities of all the tree topologies under a defined node selected for crossover. It is substantially different from the simple schema theory for linear chromosome. The advantage of unbiasedness to defining length and efficient exploration by controlling probability of a node being swapped to 0.5 cannot be captured.

EvoGraph encodes graphs in form of matrices on which evolutionary algorithms such as crossovers and mutations are applied. The process resembles GA but in a 2-dimensional matrix form instead of a linear chromosome. Each node in a graph has a probability of being swapped,  $P_0 = 0.5$  as shown in mathematical proof of the in this section. Each parent is required to divide its nodes into 2 groups. One group will be retained and the other group swapped. The edge connections within each group of nodes will be retained. EvoGraph is designed for evolving graphs in general.

#### **4.5 The EvoGraph Process**

After the development of encoding schemes for different type of graphs and the detailed mechanism of their evolution operators, graphs can be evolved like other EAs in common. There are several scheme for the selection process: roulette wheel selection and its extensions, scaling techniques, tournament, elitist models, and ranking methods [133][139]. We adopt the first selection method proposed by Holland [139], the roulette wheel selection, where individuals with higher fitness have higher probability of being selected. The evaluation function maps the solutions to a fully ordered set of positive real values, thus allowing minimization and negativity. The main steps for evolution of graphs are listed as follow. Like many other EAs, EvoGraph consists of the following steps.

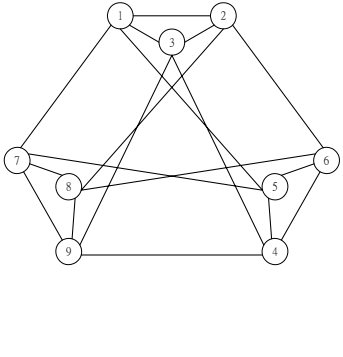
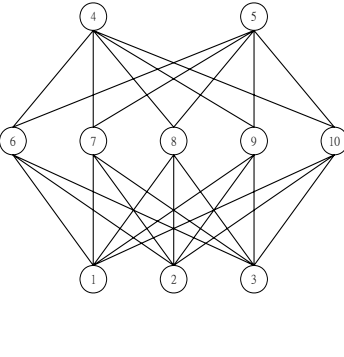
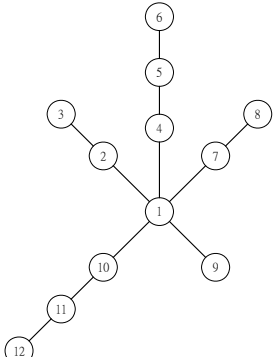
1. Given simple elements of a molecule, initialize a population of graphs at random using these elements.
2. Evaluate and assign fitness to each graph.
3. Select two graphs for reproduction using the roulette wheel selection scheme.
4. Apply crossover and mutation operators on the selected graphs.

5. Replace two the least-fit graphs in the existing population by the newly generated offspring. (Steady State Reproduction).
6. Repeat Steps 2 to 5 until the termination criteria are met.

#### **4.6 Experiments on Comparison of Performance between EvoGraph Operators and Conventional Evolution Operators**

Experiments are set up to compare the performance of EvoGraph operator and the conventional evolution operators. The two groups of operators are used to evolve the same target graphs separately and their maximum fitness and converging generations are observed. The difference between graph topologies can be measured by the edit distance between graphs. Edit distance is defined by a sequence of operations, including edge and vertex deletion and insertion, which transform one graph into another. [93] reveals that given the same number of nodes, the edit distance between two graphs bear an approximate linear relation with the Euclidean distance between their graph adjacency matrix spectrums (spectrum) as well as their Laplacian matrix spectrums (Laplacian spectrum). Making use of this relationship, we select a graph topology, an ANN bipartite graph, and a tree topology as the targets for experiments. The fitness function is designed to minimize the Euclidean distance between the spectrum or Laplacian spectrum of the evolved graph and the target graph. In order to avoid cospectral graphs (graphs having same spectrum but different topologies) being evolved, these selected graphs should be uniquely determined by their spectrums, known as *determined by spectrum* graphs or DS-graphs [94].

Three DS-graphs are selected for experiments for comparing performance of EvoGraph operators to the conventional evolution operators on evolution of graph, ANN bipartite graph and tree topology as follow. Lattice graph,  $L_k$  with  $k \neq 4$ , and regular complete bipartite graph,  $K_{mn}$  are known to be graphs uniquely determined by their spectrum [94]. Starlike trees (trees with only one node with degree higher than 2) is uniquely determined by its Laplacian spectrum [95]. The above target DS-graphs to be evolved and their corresponding spectrums are illustrated in Figure17.

		
spectrum = [-2, -2, -2, -2, 1, 1, 1, 1, 4]	Spectrum = [5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5]	Laplacian spectrum = [0, 0.16, 0.3, 0.7, 1, 1, 1, 1.54, 2.4, 3.88, 4.30, 5.72]
(a) Lattice graph $K_n$ n=3 and its spectrum	(b) ANN regular complete bipartite graph $K_{mn}$ n=5 and its spectrum	(c) Starlike tree with only degree of node '1' > 2 and its Laplacian spectrum

**Figure 17:** Target DS-graphs to be evolved with their corresponding spectrums

#### 4.6.1 Fitness Function

Formally, the evaluation function and the fitness function are distinguished from each other. The evaluation value of a solution in the population may be of any form, such as negative value. It is required to map the evaluated values of the solutions in the population to a set of positive real fitness values for maximization, ranking and selection in EA. We create an evaluation function that satisfies the fitness function requirements so that the



mapping step is not required. The evaluated values of the solutions are always positive and less than or equal to 1. They can be maximized ranked, and selected in EvoGraph according to their evaluation values.

There are several objectives to be achieved in the optimization process. There are several objectives to be matched by the solution. We combined all the target objective values to be matched under on fitness function in order to search for a solution that matches all of them with fitness value 1 or a solution that is closest to 1 in case the perfect solution cannot be found. The trade off between different objectives in arriving at the solution is not of interest. Hence, we do not adopt the principle of multi-objective optimization which creates a range of solutions at the Pareto equilibrium state leaving the selection of solution depends on the relative trade off between different objectives determined by the user. The distance between a graph in the process of evolution and the target graph to be evolved is measured by the Euclidean distance between their respective spectrum or Laplacian spectrum. The Euclidean distance is defined as follow. Let  $\mathbf{S}_A = [a_1, a_2, \dots, a_n]$  and  $\mathbf{S}_B = [b_1, b_2, \dots, b_n]$  be the spectrums or Laplacian spectrum of graph  $G_A$  and graph  $G_B$  respectively. The elements in  $\mathbf{S}_A$  and  $\mathbf{S}_B$  are eigenvalues arranged in ascending order such that  $a_1 \leq a_2 \leq \dots \leq a_n$  and  $b_1 \leq b_2 \leq \dots \leq b_m$ .

If the number of nodes of  $G_A$  and  $G_B$  are equal, the Euclidean distance between them is  $d(G_A, G_B) = \sqrt{\sum_{i=1}^n |a_i - b_i|^2}$ . If the sizes of  $G_A$  and  $G_B$  are unequal, the size of the shorter spectrum is increased by adding dimensions to bring it to match the other spectrum and zero elements are assigned to the new created spaces. The distance can then be computed with the same formula. For example, the Euclidean distance of spectrums between the graphs in Figure16(a) and (b) is

$$\sqrt{(-2-5)^2 + (-2)^2 + (-2)^2 + (-2)^2 + 1^2 + 1^2 + 1^2 + 1^2 + 4^2 + (0-5)^2} = 10.3 \quad . \quad \text{Since}$$

closeness of the number of nodes between two graphs can facilitate the minimization of Euclidean distance, our fitness function is also designed to minimize the difference in number of nodes between two graphs as follow.

$$fitness = \frac{1}{1.1^{(|\delta(D)|+|\delta(V)|)}}$$

$|\delta(D)|$  = Euclidean distance between spectrums (for lattice graph and ANN bipartite graph) or Laplacian spectrum (for starlike tree) of a graph to be evaluated and the target graph

$|\delta(V)|$  = absolute value of the difference in the number of nodes between a graph to be evaluated and the target graph

The number 1.1 is used as the base of the power in the denominator to prevent the fitness from reaching infinity when  $|\delta(D)|$  and  $|\delta(V)|$  approach zero. The maximum fitness is one. This happens when a graph is a perfect match of the target.

#### 4.6.2 Experiments and Findings

The EvoGraph operators are compared with the conventional evolution operators on evolving the lattice graph, ANN bipartite graph and starlike tree in Figure17. Conventionally EP uses only mutation operator on nodes and edges (adding or deleting nodes and edges at random) for evolving ANN. Standard GP crossover is the conventional evolution operator for trees. There is no commonly adopted evolution operator for evolving graphs. We assume EP, with node mutation and edge mutation, as the conventional operator. The performance of these conventional evolution operators are

compared with EvoGraph evolution operators, *random crossover*, *Number-of-Edge* mutation, and *Number-of Node* mutation in the experiments. To make the comparison simple, evolution strategy (self adaptation by adjusting crossover and mutation rates according to change in fitness), is not incorporated in the experiments.

The experiments are divided into three groups for the three types of graphs to be evolved in Figure 17. Each group contains 20 experiments with a subgroup of 10 experiments using EvoGraph operators and the other 10 using conventional evolution operators. Each experiment follows the process in Section 4.5. A small population of fifty graphs is generated at random in the initialization stage for experiments on evolving the lattice graph. Similarly, fifty ANN bipartite graphs and trees are generated at random for their respective groups of experiments. Based on the same initial population, comparison on performance of the EvoGraph operators and the conventional evolution operators in their respective groups can be made. The number of evolution operators used in each generation of evolution in each group of experiments is listed in Figure18. Each experiment terminates when a perfect match of the target graph topology is achieved (*fitness = 1*) or when the maximum number of generations, 5000, is reached.

Graph type	No. of exp.	Conventional Evolution Operators			EvoGraph Operators		
		Node mutation	Edge mutation	Standard GP crossover	Number-of-Node mutation	Number-of-Edge mutation	Random crossover
Lattice graph	10	0	0	0	1	1	1
	10	1	1	0	0	0	0
ANN bipartite graph	10	0	0	0	1	1	1
	10	1	1	0	0	0	0
Starlike tree	10	0	0	0	0	0	1
	10	0	0	1	0	0	0

**Figure 18:** Number of evolution operators in each generation of evolution applied in the experiments

The maximum fitnesses of the resulting graphs of the experiments and the number of generations for achieving them are summarized in Figure 19 and Figure 20 respectively. The graphs plotted on fitness vs the number of generations of evolution on the best performed experiments in respective subgroups is illustrated in Appendix 1. The ANN bipartite graph topology is a topology with bipartite constraints. Its search space is much smaller than that of the trees and graphs without specialized topologies. The search is relatively easy that both conventional evolution operators and EvoGraph operators accomplish perfect match to the target ANN bipartite graph below 400 generations. The largest search space is encountered in the search for the lattice graph amongst the set of all possible graphs without specific topologies. According to [69] the number of possible graph topologies for a graph with 9 nodes, the same number of nodes as the lattice graph, is 261,080 as indicated in Figure 16. The experiments on evolving lattice graph using conventional operators cannot achieve perfect match within 5000 generations. The use of EvoGraph operators introduces randomness to explore more novel topologies during the search as illustrated in Section 4.4. EvoGraph evolved the target lattice graph at an

average of 438 generations. Though the search space of tree topologies is smaller than that of graphs without specific topologies, the standard GP crossover cannot achieve perfect match to the starlike tree topology within 5000 generations whilst the EvoGraph *random crossover* achieves perfect match at an average of 520 generations. This demonstrates the benefit of introduction of randomness in the tree crossover process.

experiments	Lattice graph		ANN bipartite graph		Starlike tree	
	conventional evolution operators	EvoGraph operators	conventional evolution operators	EvoGraph operators	conventional evolution operators	EvoGraph operators
1	0.9147	1	1	1	0.8529	1
2	0.9147	1	1	1	0.9537	1
3	0.9104	1	1	1	0.8351	1
4	0.9482	1	1	1	0.8193	1
5	0.9091	1	1	1	0.8529	1
6	0.9091	1	1	1	0.9374	1
7	0.9091	1	1	1	0.8539	1
8	0.9091	1	1	1	0.9602	1
9	0.9091	1	1	1	0.8529	1
10	0.9142	1	1	1	0.8334	1
average	0.9148	1	1	1	0.8752	1

**Figure19:** Maximum fitness of resulting graphs generated by the experiments

experiments	Lattice graph		ANN bipartite graph		Starlike tree	
	conventional evolution operators	EvoGraph operators	conventional evolution operators	EvoGraph operators	conventional evolution operators	EvoGraph operators
1	184	202	330	136	1705	184
2	3995	657	313	212	3438	776
3	2139	430	229	177	241	119
4	851	391	221	240	984	472
5	1140	184	259	281	827	721
6	1031	439	202	189	3257	811
7	1280	825	203	115	2083	63
8	2116	191	265	205	163	818
9	434	452	423	123	984	424
10	3971	611	166	159	468	817
average	1714	438	261	184	1415	520

**Figure 20:** Generation number on reaching maximum fitness of graphs evolved in the experiments

## Evolution on Architectural Space Topology

---

Among the many possible applications, EvoGraph can be used for the automation of spatial configuration. Spatial configuration is concerned with finding feasible locations for a set of interrelated objects that meet design requirements and maximize design quality in design preferences. Spatial configuration is necessary for all physical design problems such as component packing [7], route path planning [8], VLSI [9], and architectural layout design [10]. Applications in architectural layout design are particularly interesting because, in addition to common engineering objectives such as cost and performance, it is concerned especially with aesthetics and usability, which are generally more difficult to describe formally. Also, the components in a building layout (rooms or walls, etc.) often do not have pre-defined dimensions, so that every component of the layout is resizable.

The conventional approach to architectural design is for an architect to receive a briefing from the client, usually a layperson, on functional requirements. Once such requirements are obtained, it is then up to the architect's individual skills to convert the client's requirements into building plans. The conversion process is idiosyncratic and very dependent on the artistic talent of individual architects. To facilitate the architectural layout design process so that both experienced and less experienced architects can respond quickly to their clients' requests, we show how EvoGraph can be used to perform conventional architectural layout design typically carried out manually.

## 5.1 The problem of Architectural Space Topology Design

Architectural space planning is the process of allocating a set of space elements according to certain design criteria. It usually results in topological or geometrical relationships between elements. Given a topology that describes the adjacencies between the space elements, different geometrical shapes can be allocated to a space element to satisfy the topology. The mapping of geometries to a space element in a topology is a many to one mapping. Conversely, given the geometries of the space elements it may not be feasible to retrofit them to satisfy a given topology.

There have been studies on computerized design automation of architectural space planning. [97] attempts to map the functional activities to the floor plan by encoding the functional activities in a chromosome and use an extension of GA, genetic engineering [98][100], which adopts a more aggressive approach of converting low fitness genes by high fitness genes in a chromosome in addition to the simple exchange of genes between two linear chromosomes in the conventional GA crossover operation. In [97], the architectural space topology is given and fixed. Functional activities are swapped between the architectural spaces to obtain the optimal fitness of functional activities to the spaces. This is equivalent to the situation where the architectural space topology is given in form of a graph with the spaces presented as nodes in the graph. Functional activities are node labels which are to be swapped by the genetic engineering process to fit the topology. There is no automation of the design of architectural space topology which is more fundamental.

There are studies on generating floor plans by combining some basic elements of a plan, say, a straight line of unit length, by a set of combination rules or shape grammar

[99]. The combination rules are encoded as genotype in a chromosome on which GA or genetic engineering is applied. The floor plan is generated by decoding the chromosomes as a phenotype after the crossover or mutation operation on the genotype. The fitness is evaluated according to phenotype, the floor plan. For example, in [98], a square module is used as a basic module for generation of floor plan. The modules combined to form higher level components which constitute the functional spaces of a house. The grammar of combination of the modules and the components are encoded in a chromosome on which GA is applied. The floor plan is generated according to the evolved combination rules. Like other indirect encoding method in EAs, the efficiency of evolution cannot be very high. Furthermore, the resulting geometries of the rooms are dictated by the geometries of the basic elements, such as an aggregation of squares, and the dimensions of the evolved plans are multiples of the basic elements. Hence, the creative freedom is constrained.

Some studies start off with rectangular rooms that constitute a floor plan [96][11]. [11] uses a computer program, ARCHiPLAN, as a tool to manipulate the absolute locations, orientations, and dimensions of the rectangular rooms under a set of physical constraints. During the floor plan generation process, it checks the topological consistencies of the rooms by avoiding their overlaps. [96] represents the rectangular rooms as nodes of a tree. The topology of the tree is a presentation of the room adjacencies. The tree topology and the absolute locations of the rooms are encoded in different chromosomes and each of them undergoes asexual crossovers and mutations independently. The separate results decoded from the chromosomes are put together in the floor plan. This is equivalent to the swapping of rectangular rooms attached to a tree topology where dimensions and



orientation of the rooms as well as the tree topology are changing at the same time. The efficiency of search is hindered by the combinatorial complexity. Furthermore, tree is not a good presentation of architectural space topology because it only presents the adjacencies between the nodes or rooms from one layer of node in the tree to the layer immediately above and below it. It cannot show the adjacencies between rooms in the same layer or nodes separated by more than one layer of nodes in the tree. The result is that the search tends to fixate at the first feasible design. Both [96] and [11] are designed for rectangular room plan manipulation but not other geometric shapes. The creative freedom is again constrained.

To summarize, the current studies on computerized architectural space planning concentrate on packing rooms with fixed geometries and evolving floor plans from basic elements with fixed dimensions. Architectural space topology is playing a secondary role as a condition to be satisfied. The creative freedom of the architect is limited to a certain extent by the pre-determined geometric shapes and dimensions of the basic elements adopted in the design automation algorithms. The resulting floor plans tend to be predictable and mundane. In order not to restrain the creativity of the architect, we propose to start off the design process by evolving the optimal architectural space topology while leaving the insertion of geometry of rooms to the creative hands of the architect. The most natural form of presentation of architectural space topology is in the form of a graph and we need an EA to apply on it.

Using the graph adjacency matrix, the crossover and mutation operators defined in Chapter 4, EvoGraph can be used to tackle problems that can be formulated as graphs. One such problem is that of architectural layout design. At the design inception stage in a

typical architectural project, the architect has to interpret requirements from a client who is, in most cases, a layperson. The client usually describes his or her preferred spatial groupings verbally. Such preferences can be translated into an *adjacency preference matrix*, which describes the preferred adjacency between functional areas. In Figure 21, we give an example of an adjacency preference matrix of a house, drawn up based on the requirements of a client. The numbers, which make up an *Adjacency Preference Scale* (APS), are defined in such a way that -2 means that the adjacency arrangement is *very much not preferred*, -1 means that it is *not preferred*, 1 means that it is *preferred*, 2 means that it is *very much preferred*, and 3 means that it is *extremely preferable*.

	1. SA	2. MA	3. BED	4. LR	5. D&K	6. B	7. CP	8. P	9. CIR	10. EXT
1. Study area (SA)	-2	2	1	-2	-2	-2	-2	-1	3	3
2. Master ensuite (ME)	2	-2	2	-2	-2	-2	-2	3	3	3
3. Bedroom (BED)	1	2	2	-2	-2	3	-2	1	3	3
4. Living room (LR)	-2	-2	-2	-2	1	2	1	3	3	3
5. Dining & kitchen (D&K)	-2	-2	-2	1	-2	2	-2	1	3	2
6. Bathroom (B)	-2	-2	3	2	2	3	-2	-2	3	1
7. Car park (CP)	-2	-2	-2	1	-2	-2	-2	-2	3	3
8. Patio (P)	-1	3	1	3	1	-2	-2	-2	-2	3
9. Hall/stair/circulation (CIR)	3	3	3	3	3	3	3	-2	3	-2
10. Exterior (EXT)	3	3	3	3	2	1	3	3	-2	3

**Figure 21:** Adjacency Preference Matrix

The spatial design requirements as shown in the matrix in Figure 21 can be summarized, in a more descriptive way, as follows:

A. Bedrooms should be grouped . (APS(2,3)=2).

B. Bathroom is to be shared by the bedroom, living room, and dining room.

(APS(6,3)=3, APS(6,4)=2, APS(6,5)=2).

C. Patio is to be shared by master ensuite and living room and if possible, the dining room and bedroom. It is preferable to be exposed.

( $APS(8,2)=3$ ,  $APS(8,3)=1$ ,  $APS(8,4)=3$ ,  $APS(8,5)=1$ ,  $APS(8,10)=3$ ).

D. Circulation areas to link all rooms and carpark.

( $APS(9,i)=3$  where  $1 \leq i \leq 10$  and  $i \neq 8,10$ ).

E. Study area to be attached to master ensuite. ( $APS(1,2)=2$ ).

F. All rooms, carpark, and patio are preferred to be in contact with the exterior.

( $APS(10,i)=3$  where  $1 \leq i \leq 10$  and  $i \neq 9$ ).

The approach to architectural layout design is divided into two parts: topology and geometry. Topology refers to the logical relation between layout components. Geometry refers to the position and size of each component layout. Topological decisions define constraints for the geometric design space. For example, a topological decision that “room  $i$  is adjacent to room  $j$ ” restricts the geometric coordinates of room  $i$  relative to room  $j$ . Such decisions are important and have to be made before finalizing on the geometry. What a designer needs to do is therefore to try, as much as possible, to enumerate all topologies that can produce feasible geometries [11] and then review them to select those to explore geometrically. This process is slow and time-consuming. As it is very difficult for designers, especially inexperienced ones, to run through the process of optimizing topologies manually, subsequent geometric designs produced may not best match the client’s need.

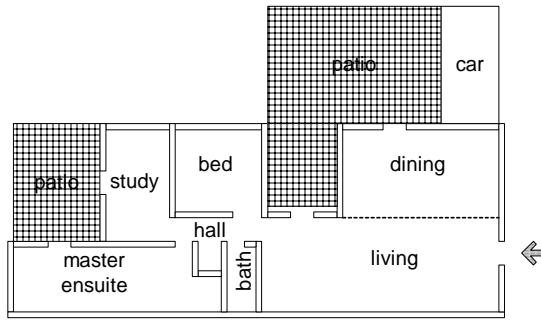
To overcome this problem, EvoGraph can be used to find possible optimal topologies in the design process so that, instead of having to deal with too many feasible but

suboptimal topologies, designers need only deal with the optimized ones when starting geometry design.

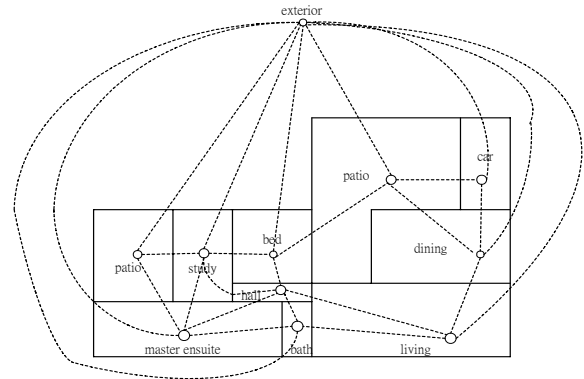
## **5.2 Conversion Between Floor Plan and Graph Representing Architectural Space**

### **Topology**

A floor plan can be converted into a graph representing its architectural space topology and vice versa. EvoGraph optimizes graphs representing different floor plans but not the floor plan themselves. The optimized graph achieved by EvoGraph has to be converted to a floor plan for use. Before we discuss the evolution process, we introduce the conversion between floor plans and the graph representing it. We represent an individual architectural space by a node in a graph. If two architectural spaces, represented by two nodes, share a common boundary, say a wall, we represent this adjacency by an edge connecting the two nodes. Figure 22–23 give an example of the floor plan of a house with respective functional rooms and its corresponding graph representation on its architectural space topology. The graph in Figure 23 should be a planar graph which can be encoded in a graph adjacency matrix. EvoGraph conducts evolution on the graph adjacency matrices to obtain the optimal graph topology. Floor plans can then be drawn up by decoding the resultant optimal graph adjacency matrix.



**Figure 22:** The functional area floor plan of a house



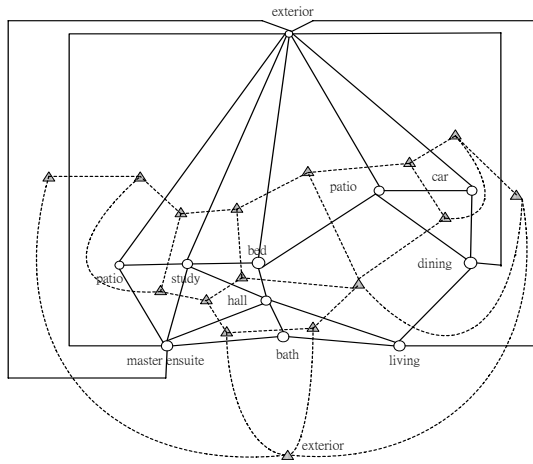
**Figure 23:** Graph (dotted line and circular node) representing architectural space topology of the floor plan in Figure 22

For decoding, given a graph adjacency matrix, the corresponding graph is drawn. This graph may represent a one storey or a multi-storey building. It can be decomposed into one or more planar subgraphs. Each of the planar subgraphs represents a floor plan of a storey. There can be more than one way of decomposition of a graph representing a multi-storey building into planar subgraphs representing individual storey. The way of decomposition depends is the architect's choice. This is equivalent to the architect's juggling with an optimized bubble diagrams in the conceptual design stage. An example on decomposition of a graph into several planar subgraphs is demonstrated in Section 5.5.3 by using one of the optimal architectural space topologies generated by the experiments. In the following, we show how a planar graph is converted into a space enclosure representing a floor plan.

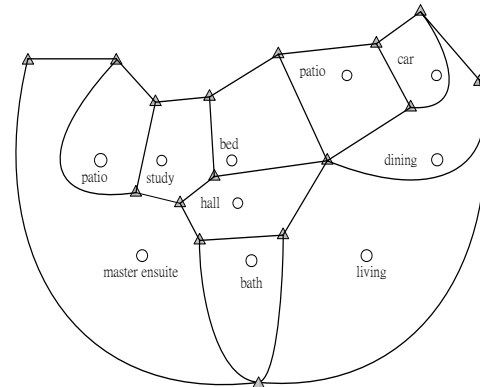
Given a planar subgraph that represents the architectural space topology of a floor plan, its dual graph (which is also planar) can be drawn up in two steps illustrated by an example in Figure 24-25 as follows.

1. Lay out the planar graph on a plane without crossing edges (shown as solid lines and circular nodes in Figure 24).
2. Every edge in the planar graph represents the boundary separating the two spaces corresponding to the nodes it connects. In order to construct the dual graph, a pair of nodes is created on both sides of an edge in the planar graph (shown as triangular-shape nodes in Figure 24). A new edge is then created to connect the newly created nodes (shown as a dotted line in Figure 24). These new edges are the spatial boundaries of the architectural space topology in the planar graph. The dual graph in Figure 25 can then be derived from Figure 24.

In this example the adjacencies between functional spaces in Figure 24 are the same as the floor plan in Figure 22. The conversion of the dual graph in Figure 25 into the floor plan in Figure 22 is done by inserting geometries into the space enclosed by the edges in the dual graph.



**Figure 24:** Dual Graph (dotted lines and triangular nodes) derived from the solid line graph



**Figure 25:** Functional Spaces enclosed by edges of Dual Graph (solid line and circular nodes)

### 5.3 Additional Mutation Operators

Given an Architectural Space Topology represented by a graph, the mutation of room functional spaces represented by nodes in the graph is helpful to improve the total APS. In order to facilitate this kind of mutation, two mutation operators are developed in addition to the two basic mutation operators introduced in Section 4. They are *Node-Label* mutation and *Swap-Node* mutation.

#### 5.3.1 Node-Label Mutation

The *Node-Label* mutation operator allows us to replace one node by another in the same graph. It works by selecting a node to be replaced and a node to replace it at random as follow.

1. For a graph  $G^P(V^P, E^P)$  with node set  $\{v_1^P, v_2^P, \dots, v_i^P, \dots, v_j^P, \dots, v_n^P\}$  we construct

its corresponding adjacency matrices as  $\mathbf{G}^P$ .

2. Select a node in the graph  $G^P$  and corresponding adjacency matrix  $\mathbf{G}^P$  randomly.

Assume that the node chosen in  $\mathbf{G}^P$  are  $v_i^P$ .

3. Replace node label  $v_i^P$  by another node label chosen at random in the same graph, say,

node label  $v_j^P$ , to form children graph  $G^C$  and adjacency matrix  $\mathbf{G}^C$ . The order of

node set in  $\mathbf{G}^C$  is changed to  $v_1^P, v_2^P, \dots, v_j^P, \dots, v_j^P, \dots, v_n^P$ .

We give an example of the *Node-Label* mutation operator in Figure 26 below.

		$\mathbf{G}^P$							
	A	B	C	D	E	F	G	H	
A	-	1	0	0	1	0	0	0	
B		-	1	1	0	0	0	0	
C			-	1	0	0	0	1	
D				-	1	0	0	0	
E					-	1	0	0	
F						-	1	0	
G							-	0	
H								-	

(a) Step 1. Select C to be replaced by F

		$\mathbf{G}^C$							
	A	B	F	D	E	F	G	H	
A	-	1	0	0	1	0	0	0	
B		-	1	1	0	0	0	0	
F			-	1	0	0	0	1	
D				-	1	0	0	0	
E					-	1	0	0	
F						-	1	0	
G							-	0	
H								-	

(b) Step 2. Replace C with F

**Figure 26:** The Node-Label Mutation operator illustrated

### 5.3.2 Swap-Node Mutation

The *Swap-Node* mutation operator allows us to swap two nodes in the same graph. It selects a pair of nodes at random and then swaps them as follows.

1. For a graph  $G^P(V^P, E^P)$  with node set  $\{v_1^P, v_2^P, \dots, v_i^P, \dots, v_j^P, \dots, v_n^P\}$  we construct its

corresponding adjacency matrices as  $\mathbf{G}^P$ .

2. Two nodes in the graph  $G^P$  are selected and this is done by choosing 2 nodes in  $\mathbf{G}^P$



randomly. Assume that the nodes chosen in  $\mathbf{G}^P$  are  $v_i^P$  and  $v_j^P$ .

- Swap  $v_i^P$  and  $v_j^P$  to form children graph  $G^C$  and adjacency matrix  $\mathbf{G}^C$ . The order of node set in  $\mathbf{G}^C$  is changed to  $v_1^P, v_2^P, \dots, v_j^P, \dots, v_i^P, \dots, v_n^P$ .

We give an example of the *Swap-Node* mutation operator in Figure 27 below.

$\mathbf{G}^P$									$\mathbf{G}^C$								
	A	B	C	D	E	F	G	H		A	B	F	D	E	C	G	H
A	-	1	0	0	1	0	0	0	A	-	1	0	0	1	0	0	0
B		-	1	1	0	0	0	0	B		-	1	1	0	0	0	0
C			-	1	0	0	0	1	F			-	1	0	0	0	1
D				-	1	0	0	0	D				-	1	0	0	0
E					-	1	0	0	E					-	1	0	0
F						-	1	0	C						-	1	0
G							-	0	G							-	0
H								-	H								-

(a) Select two nodes F and C to be swapped

(b) Swap the nodes F and C.

**Figure 27:** The Swap-Node mutation operator illustrated

## 5.4 Fitness Function

Given that the objective is to find optimal architectural topological designs, we propose to use a fitness function in the evolutionary process that takes into consideration:

- clients' preferences as given in an Adjacency Preference Matrix and
- physical constraints as given by an *Adjacency Limitation* defined to be the maximum number of adjacent rooms that one room can be in contact with. This *Adjacency Limitation* can be expressed as the *valence* of a node in a graph representation,
- budget constraint,
- the range of relative ratios between rooms and
- the minimum functions that are required to constitute an acceptable design. The adjacency preference is quantified in Figure 21. The maximum valence of a node, the cost of providing each node are given in

Table 2(a), and the acceptable range of relative room ratios are given in Table 2(b). The 9 different functions in Figure 21 should be included in each design.

The lowest range of budget allowed for the experiments is 30 to 34. The second range is 35 to 39 and so forth until the budget reaches the range from 55 to 59. EvoGraph is applied to search for the topologies that maximize the adjacency scale within the budget range and the valance constraints of individual nodes. The fitness function is defined as follow.

$$fitness = \frac{x}{2^{a+b+c+d}} \quad (1)$$

where  $x$  = sum of the Adjacency Preference Scales (APS)

$a$  = absolute deviation to the budget range

$b$  = absolute value of sum of excess valance of nodes

$c$  = total absolute deviation to the allowed range of room ratios

$d$  = number of functions deficient in the graph

In other words, the APS is the value to be maximized and the deviations to the other assigned constraints are the value to be minimized. The denominator is made an exponential function to eliminate the mathematical indeterminate case when both factors are equal to zero. When all the imposed constraints are satisfied, the values of  $a$ ,  $b$ ,  $c$  and  $d$  will be 0. As such, the fitness value equals APS of the graph topology.

Once terminated, the optimal graph can be decoded and the floor plans can be drawn as described in Section 5.2. Using the optimal architectural space topology given in a form like Figure 25, an architect can insert his or her favorite architectural motifs to complete the design.

Label	Functional Space	Max. Valence	Cost
1	Study area	4	3
2	Master ensuite	4	4
3	Bedroom	4	3
4	Living room	4	6
5	Dining and kitchen	4	5
6	Bathroom	4	2
7	Carpark	No limit	1
8	Patio	No limit	4
9	Hall/stair/circulation area	6	4
10	Exterior	No limit	0

**Table 2(a):** The function space description corresponding to the labels

Function 1/Function 2	Min. Ratio	Max. Ratio
Study area/Master ensuite	0	1
Study area/Bedroom	0	1
Study area/Patio	0	4
Master ensuite/Patio	1	2
Bedroom/Living room	1	3
Bedroom/Bathroom	1	2
Bedroom/Patio	1	4
Living room/Bathroom	1	2
Living room/Patio	1	2

**Table 2(b):** The Relative Room Ratios

The success of achieving the design requirements A to F by the evolution process depends on their corresponding values of APS in Figure 21 and the values of constraints in Table 2(a) and (b). If they are in line with each other, the probability of achieving the requirement is high. For example, the ‘hall /stair/ circulation’ in has a high valence of 6 as shown in Table 2(a). In combination with its high APS to other functional spaces, except patio and exterior, in Figure 21, requirement D has a high probability of success as

the constraint is in line with the design requirement. Another example is that there is no limit to the adjacency of ‘carpark’, ‘patio’ and ‘exterior’. In combination with the zero cost for ‘exterior’, it is expected that the resulting design will favor also design requirement F. Requirement D is a ‘closed approach’ to design with internal circulation area linking all the function spaces while requirement F is an ‘open approach’ to the with functional spaces facing the exterior. These conflicting requirements may coexist in the optimal solution in each of the experiments. We explore the relative presence of requirements D and F in relation to the cost efficiency (defined in terms of APS achieved/unit cost) in the evolved solution under the same set of constraints in different experiments. The results are analysed in Section 5.5.2.

## **5.5 Experiments**

The objective of the experiments we performed is to determine if EvoGraph can be used to effectively find optimal architectural topological design. For this purpose, we assume that we need to design a house with 9 functional spaces with an adjacency preference matrix as shown in Figure 21, and each functional space has other constraints specified in Table 2(a) and 2(b). For our experiments, the fitness function given by Equation (1) is used.

### **5.5.1 Initialization and EvoGraph Parameters Selection**

An initial population of 100 graph adjacency matrices is generated at random. The nodes encoded in each of them are given the same labels as that in Figure 21.

As described in Chapter 4, the approach adopted is a general evolutionary algorithm for graphs that is not biased toward any preferred topologies. The crossover and mutations operators selected are also unbiased, they are *random crossover*, *Number-of-Edge* mutation, *Number-of-Node* mutation, *Node-Label* mutation and *Swap-Node* mutation. To place emphasis on exploration of search space to exploitation, mutations are having the same probability as crossover as shown in Table 3.

Operator	Probabilities of being selected
Random crossover	0.2
Number-of-edge mutation	0.2
Number-of-node mutation	0.2
Node-label mutation	0.2
Swap-node mutation	0.2

**Table 3:** The probabilities of each operator being selected for reproduction

### 5.5.2 Experimental Results and Findings

Using the same parameters described above, we run experiments with a population size of 100 graph adjacency matrices according to the process in Section 4.5. The maximum number of generation is 5,000. Convergence is considered to be reached when the maximum fitness has been stagnant for 2000 generations. The process is stated as follow.

In order to compare the performances of EvoGraph on the same basis, all the experiments start with the same initial population. Experiments on 6 budget ranges are carried out each having a budget interval of 5 starting from the lowest range at 30 to 34 (Experiment 1) and finish at its double at 55 to 59 (Experiment 6). Each experiment is run 10 times and the one with highest APS is adopted. All the other constraints on budget,

valence of nodes, minimum functions required, and range of room ratios should be satisfied in order to be acceptable as a valid solution. In such circumstances, the fitness value according to (1) should be equal to the total APS of the resulting graph. The graphs indicating maximum fitness change with respect to the number of generations are illustrated in Appendix 2.

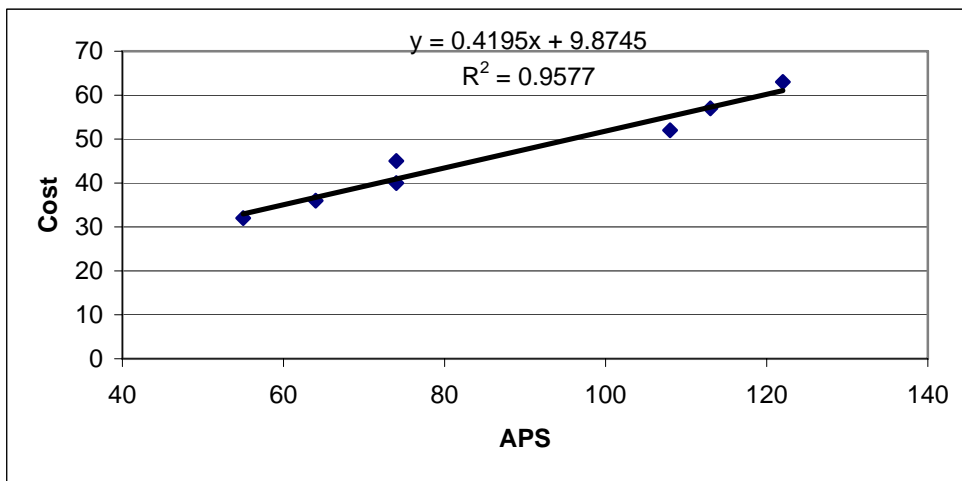
Optimal architectural space topologies are generated by the experiments in the form of adjacency matrices. Experiment 5 has the highest APS relative to cost. The adjacency matrix with APS value generated by experiment 5 is illustrated as an example in Figure 33. When there is more than one node having the same function, a small letter will be added to the node number label to differentiate them. For example, two different ‘circulation areas’ will be represented by ‘9a’ and ‘9b’ etc.. All the adjacency matrices with corresponding APS that represent the optimal architectural space topologies generated by the experiments are included in Appendix 2. The fitness plots throughout the evolutions are also included.

The summary of experimental results on the properties of the graphs generated with respect to the number of nodes, converging generation, APS and cost are summarized in Table 4. Note that all results satisfy the constraints on valence and room ratios in Table 2.

<b>Experiment</b>	<b>Budget Range</b>	<b>No. of Nodes</b>	<b>Converging Generation</b>	<b>APS</b>	<b>Cost</b>	<b>APS/Cost</b>
1	30-34	10	1312	55	32	1.719
2	35-39	11	611	64	36	1.778
3	40-44	12	1536	74	40	1.85
4	45-49	13	1036	74	45	1.644
5	50-54	15	1294	108	52	2.077
6	55-59	16	2101	113	57	1.982

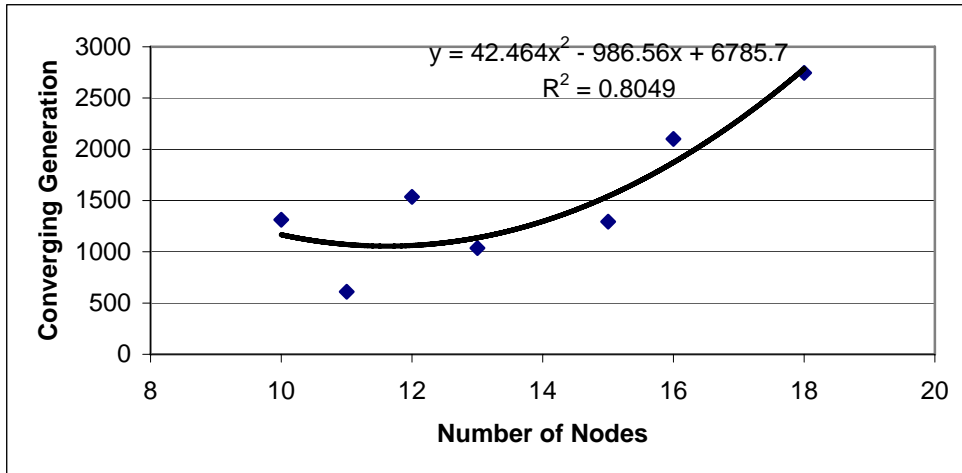
**Table 4:** Summary of Experiment Results

The relation between cost and APS is indicated in Figure 28. It is close to linear relation with least square regression value close to 1 at 0.9577. This reflects the fact that the more the client pays, the more satisfaction he can derive from the design. The marginal increase in APS per unit increase in cost is approximately 0.42. The number of functional space in a house (number of nodes in the resulting graph) also increases as the budget increase at a rate of one additional functional space per 5 units of budget increase.



**Figure 28:** Relation between Cost and Adjacency Preference Scale (APS)

The relation between the converging generation and the number of nodes is indicated in Figure 29. It approximates a parabolic relation with square regression value 0.8049. The converging generation is proportional to  $\sqrt{V}$ .



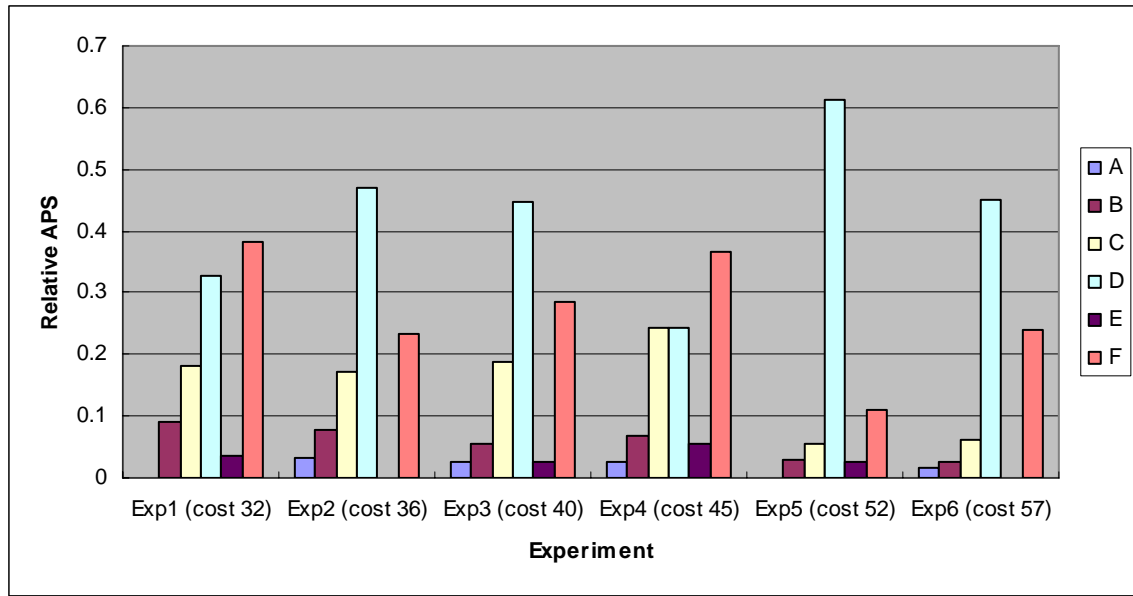
**Figure 29:** Relation between number of nodes and converging generation

The APS attained by each experiment with respect to the design preferences stated in Section 5.1 are divided by the total APS of the same optimal graph to observe the contribution of the design objective to the total APS. It is defined as relative APS and shown in Table 5. The total contribution of the 6 experiments to each design objective is calculated and ranked.

	Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Total
A. Bedrooms should be grouped	0	0.0313	0.027	0.027	0	0.0177	0.103
B. Bathroom is to be shared by the bedroom, living room, and dining room	0.0909	0.0781	0.0541	0.0676	0.0278	0.0265	0.345
C. Patio is to be shared by master ensuite and living room and if possible, the dining room and bedroom. It is preferable to be exposed	0.1818	0.1719	0.1892	0.2432	0.0556	0.0619	0.9036
D. Circulation areas to link all rooms and carpark	0.3273	0.4688	0.4459	0.2432	0.6111	0.4513	2.5476
E. Study area to be attached to master ensuite	0.0364	0	0.027	0.0541	0.0267	0	0.1442
F. All rooms, carpark, and patio are preferred to be in contact with the exterior	0.3818	0.2344	0.2838	0.3649	0.1111	0.2389	1.6149

**Table 5:** Relative APS of Experiments





**Figure 30:** Histogram of experimental results

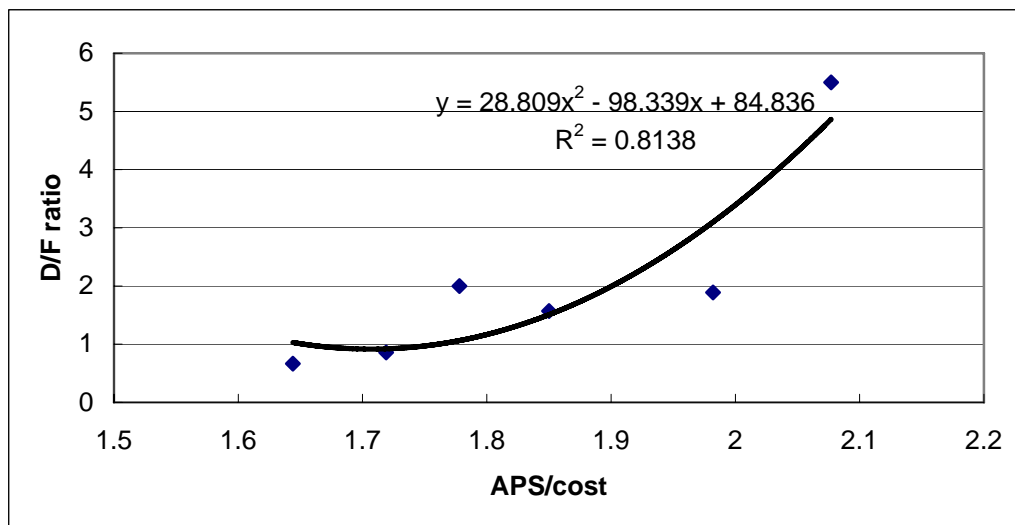
The budget increases from experiment 1 to experiment 6. Referring to Figure 30, property A and E are relatively insignificant. Property B tends to decrease when budget increases. There is a sudden drop in property C when cost increases from 45 to 52 with a corresponding upward trend in property D. The general trend of design objectives achieved by all the experiments is arranged according to the magnitude of the total relative APS in Table 5. They are listed in descending order of relative APS as follow.

1. Circulation areas to link all rooms and carpark.
2. All rooms, carpark, and patio are preferred to be in contact with the exterior.
3. Patio is to be shared by master ensuite and living room and if possible, the dining room and bedroom. It is preferable to be exposed.
4. Bathroom is to be shared by the bedroom, living room, and dining room.
5. Study area to be attached to master ensuite.
6. Bedrooms should be grouped.

Design objectives D and F are amongst the highest in the rank. They govern if the building design is open with the exposed patio unifying all the other functions (open approach) or a closed one with the internal circulation space as a major source of linkage (closed approach). The experiments reveal that both type of design are achievable. Experiments 1 and 4 have relative APS of property F higher than D (open approach) whilst the rest of the experiments reveal the opposite (closed approach). Experiment 5 has the highest property D and the lowest property F and it is the most cost efficient solution as indicated in Table 4. There is a trend of increasing cost efficiency with ratio of relative APS of property D to property F as illustrated in Table 6 and Figure 31.

Experiment	APS/Cost	D/F ratio
1	1.719	0.8573
2	1.778	2
3	1.85	1.5712
4	1.644	0.6665
5	2.077	5.5005
6	1.982	1.8891

**Table 6:** Cost efficiency in relation to relative APS ratio of property D to F



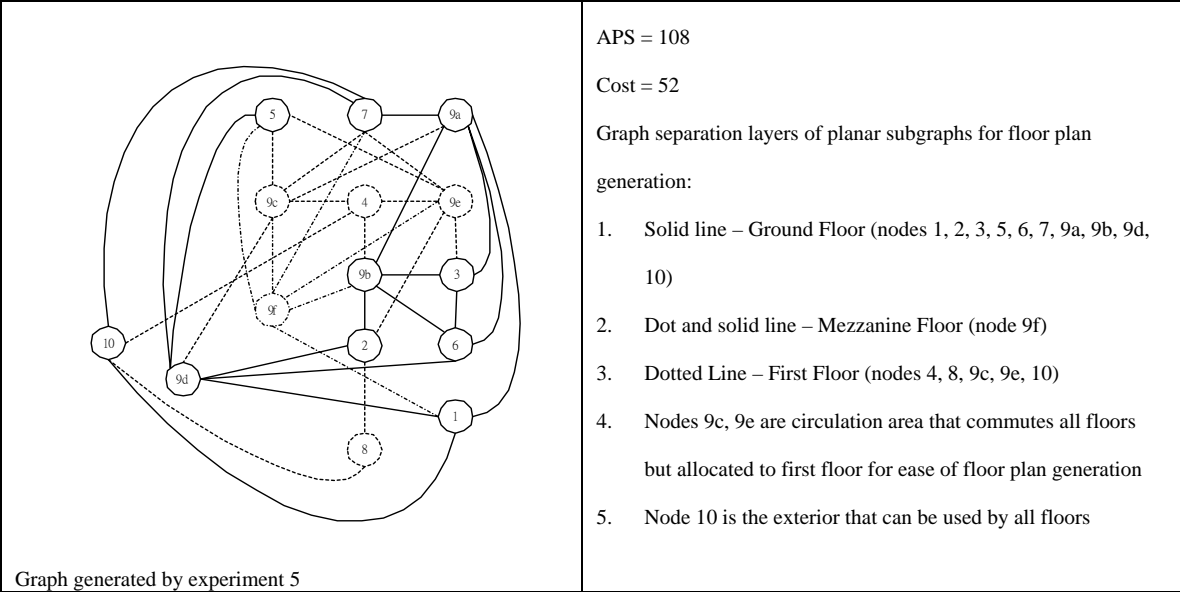
**Figure 31:** Cost efficiency vs relative APS ratio of property D to F

### 5.5.3 Optimal Architectural Space Topology Conversion to Floor Plans

The maximum fitness value of the experiments we performed above range from 55 to 113. Using the approach given in Section 5.2, each of these optimal graphs can be converted to topological space and then to a floor plan. Note that each architectural space topology may have more than one geometrical floor plan corresponding to it. The most cost efficient solution generated by experiment 5 is illustrated below as an example. The following illustration is only one of the many geometrical plans used to interpret the optimal architectural space topology. When there are overlapping edges in a resulting graph, it is separated into layers without overlap to represent different floors (differentiate by solid and broken edges and nodes in the Figure 33).

	9a	9b	6	3	9c	4	2	9d	7	9e	5	9f	8	1	10
9a	-	3	3	3	3	0	0	0	3	0	0	0	0	3	0
9b		-	3	3	0	3	3	0	0	0	0	3	0	0	0
6			-	3	0	0	0	3	0	0	0	0	0	0	0
3				-	0	0	0	0	0	3	0	0	0	0	0
9c					-	3	0	3	3	0	3	3	0	0	0
4						-	0	0	0	3	0	0	0	0	3
2							-	3	0	3	0	0	3	0	0
9d								-	3	0	3	0	0	3	0
7									-	3	0	3	0	0	3
9e										-	3	3	0	0	0
5											-	3	0	0	0
9f												-	0	3	0
8													-	0	3
1														-	3
10															-

**Figure 32:** Adjacency matrix with APS generated by experiment 5

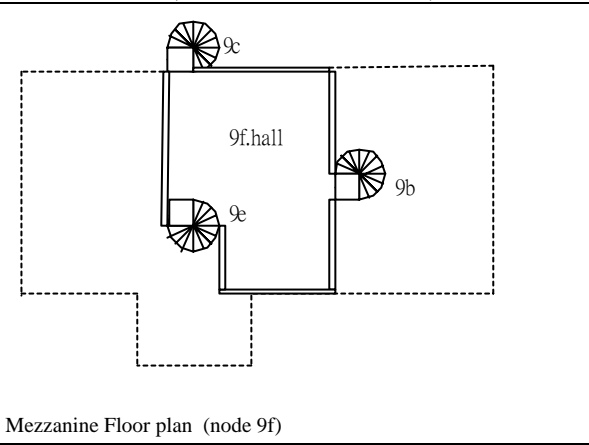
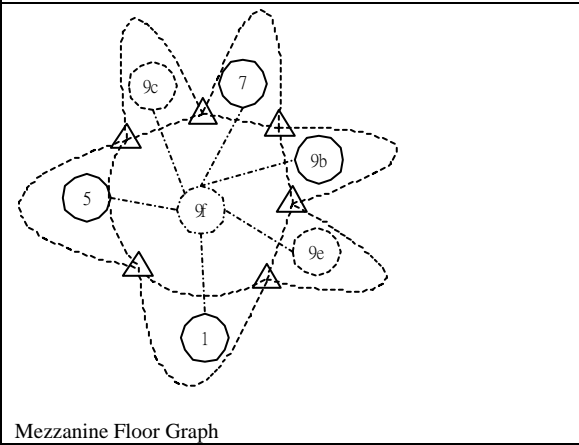
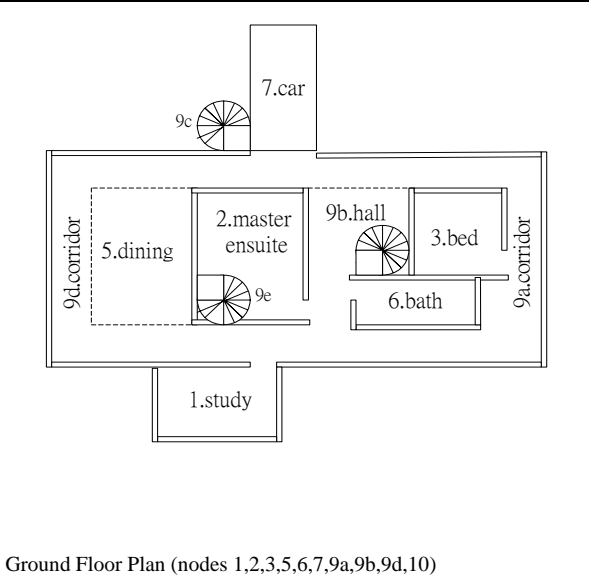
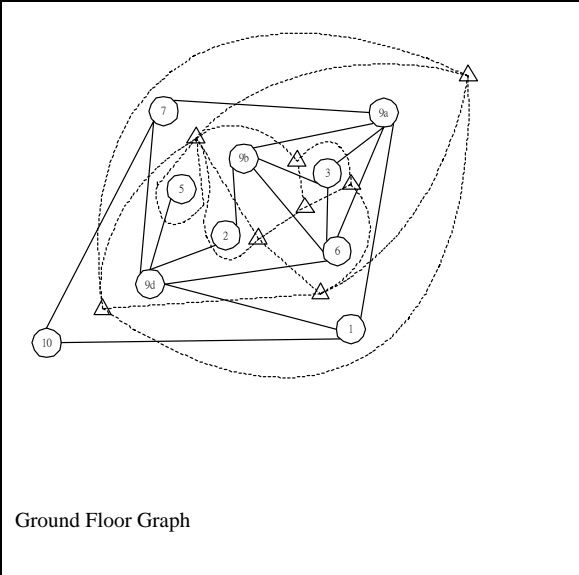


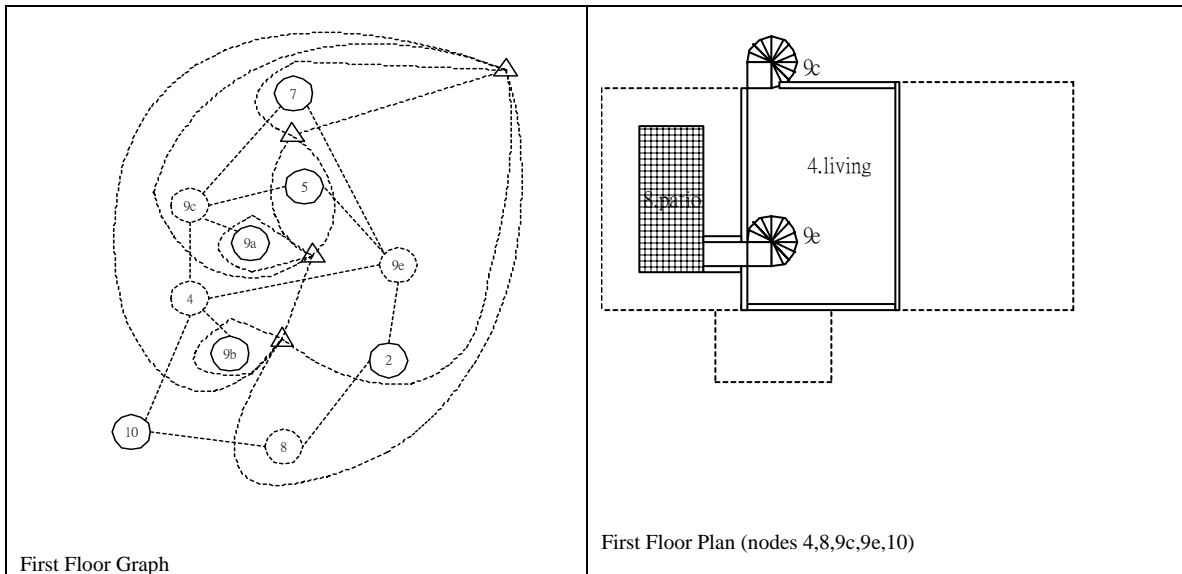
APS = 108

Cost = 52

Graph separation layers of planar subgraphs for floor plan generation:

1. Solid line – Ground Floor (nodes 1, 2, 3, 5, 6, 7, 9a, 9b, 9d, 10)
2. Dot and solid line – Mezzanine Floor (node 9f)
3. Dotted Line – First Floor (nodes 4, 8, 9c, 9e, 10)
4. Nodes 9c, 9e are circulation area that commutes all floors but allocated to first floor for ease of floor plan generation
5. Node 10 is the exterior that can be used by all floors





**Figure 33:** Optimal Architectural Space Topology generated by experiment 5 and corresponding floor plans

This research is an attempt to explore the application of EvoGraph on architectural space topology generation. Using 10 nodes that include 9 functional areas and one node representing the exterior, EvoGraph generates optimal design topologies that satisfy budget requirements and other design constraints. These alternatives compete with each other by compromising design objectives in give-and-take situations and eventually arrive at the one that best fits the design intent expressed in the fitness function.

---

## Evolution on Space Frame Topology and Geometry

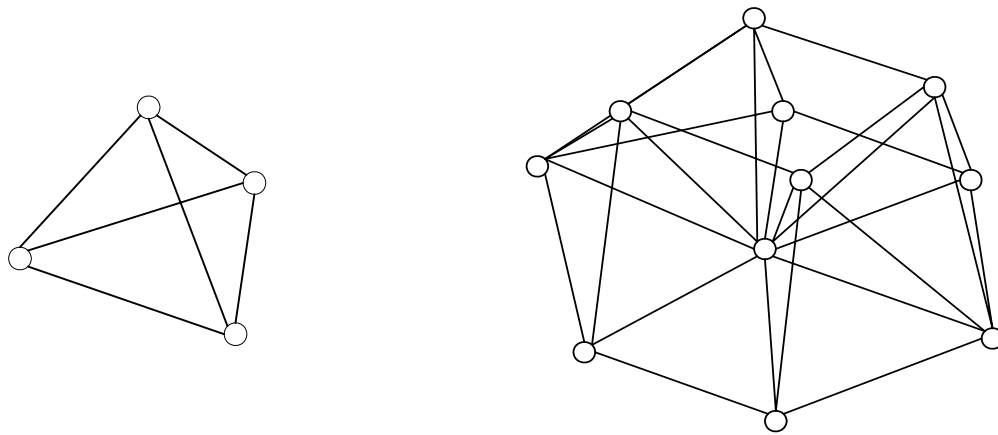
---

Space frame has been widely adopted as an aesthetic design feature in architecture. A simple structure space frame can be used as decorative features such as lanterns and hanging features in shopping malls. Repetition of a simple modular space frame structure can form large space frame that constitutes a major element of a building such as roof cover, canopy, external wall and even the whole building enclosure. The process of artistic creation of a space frame with resource constraint has always been a puzzle to architects and designers. Although there are some standard proprietary space frame systems with associated software to assist design, the varieties available for choosing are limited and they are dominated by the manufacturers. The tool available for original creation of space frame modules is lacking. A hybrid solution on EvoGraph and GA is proposed in this Chapter to provide a tool that rapidly creates a space frame module design within the resource constraint available to the architect.

### 6.1 Space Frame Module Design

Space frames are designed in many forms. For example, one of the most popular basic modules of a space frame is a tetrahedron as shown in Figure 34(a). This basic module can be repeated to form a larger space frame. Six repeated modules are shown in Figure 34(b). One of the general properties of a space frame module is symmetry of geometrical shape. It serves the purpose of ease of combining many modules to form a large space frame. The symmetry of geometry can be achieved by designing space frame

modules with symmetric topology and geometry. The purpose of this chapter is to use EvoGraph to generate symmetrical space frame module topology and then use GA to search for the symmetrical geometric properties of the symmetric space frame module topology. The use of EvoGraph and GA in the evolution process forms a hybrid solution to the problem.



(a) Space Frame Module

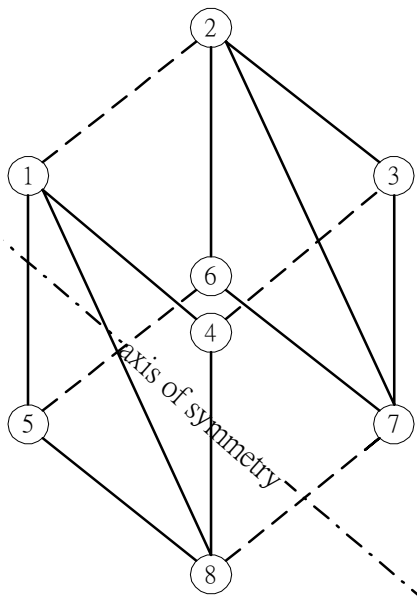
(b) Six Repeated Modules

**Figure 34:** Space Frame Module repetition illustrated

### 6.1.1 Symmetric Space Frame Module Topology Design

Topology is the first thing to be tackled in designing a space frame module because without which no geometrical form can be created. The target of this section is to search for a symmetric topology for the space frame module to provide a base for the creation of symmetric geometry. One of the properties of a symmetric space frame module is that it can be divided along its axis of symmetry into a pair of subframes which are mirror image to each other, known as ‘mirror subframes’. The remaining pair of subframes connecting the ‘mirror subframes’ is also mirror image to each other. This property can

be reflected in the adjacency matrix of the symmetric space frame module. This is illustrated by an example of a cubic space frame module with even number of nodes in Figure 35 and a pyramid with a pentagon base in Figure 36.



	1	4	8	5	2	3	7	6
1	0	1	1	1	1	0	0	0
4	1	0	1	0	0	1	0	0
8	1	1	0	1	0	0	1	0
5	1	0	1	0	0	0	0	1
2	1	0	0	0	0	1	1	1
3	0	1	0	0	1	0	1	0
7	0	0	1	0	1	1	0	1
6	0	0	0	1	1	0	1	0

(a) Dividing a space frame module into a pairs of mirror subframes {1,4,8,5} and {2,3,7,6}

(b) Partition adjacency matrix into four submatrices corresponding to the divided subframes

**Figure 35:** Example of a Cubical Symmetric Space Frame Module divided into subframes which are mirror images to each other

In Figure 35(a), the space frame with node set {1, 2, 3, 4, 5, 6, 7, 8} can be subdivided into two mirror subframes with node sets {1, 4, 8, 5} and {2, 3, 7, 6} by deleting edges connecting the two mirror subframes (dotted lines in the figure). The adjacency matrix of space frame module in Figure 35(a) is illustrated in Figure 35(b). It can be constructed in such a way that the upper left quadrant submatrix (A) that represents subframe {1, 4, 8, 5} is identical to the lower right quadrant submatrix (C) that represents subframe {2, 3, 7, 6}, or  $A=C$ . The pair of subframes connecting the pair of



mirror subframes represented by submatrices  $A$  and  $C$  is also mirror image to each other. Hence, the upper right quadrant submatrix ( $B$ ) that represents a subframe that connects the mirror subframes  $\{1, 4, 8, 5\}$  and  $\{2, 3, 7, 6\}$  is mirror image to the lower left quadrant submatrix ( $D$ ) which represents another subframe connecting the same, or  $B=D$ . The degree of symmetry of a space frame can be derived by comparing the submatrices.

Define

$N(X \cap Y)$  = number of corresponding slots having ‘1’ in both matrices  $X$  and  $Y$

$N(X \cup Y)$  = number of corresponding slots having ‘1’ in either matrices  $X$  or  $Y$

Define correlation coefficient between two adjacency matrices  $X$  and  $Y$  as

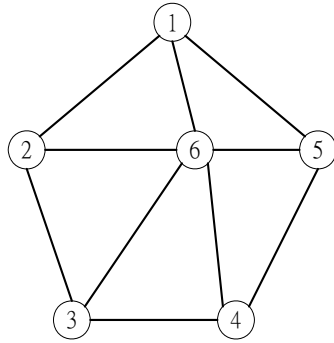
$$c^{XY} = \frac{N(X \cap Y)}{N(X \cup Y)} \quad 0 \leq c^{XY} \leq 1 \quad (1)$$

For the example of adjacency matrix for a space frame that is divided into four equal quadrants  $A, B, C, D$  shown in Figure 35(b), the higher the value of  $c^{AC}$  and  $c^{BD}$ , the more likely the space frame module is divided into two pairs of mirror subframes and hence the higher the degree of symmetry. Both  $c^{AC}$  and  $c^{BD}$  should be maximized in the EvoGraph fitness function. In Figure 35(b) and (b),  $c^{AC} = c^{BD} = 1$ .

EvoGraph is designed to work on adjacency matrices on which space frame module symmetry is encoded. The symmetry of the space frame module is encoded in the two pairs of identical submatrices diagonally opposite to each other as shown in Figure 35(b). This requires the number of nodes in the space frame module to be even and they can be subdivided into two groups with equal number of nodes by deleting edges in the space

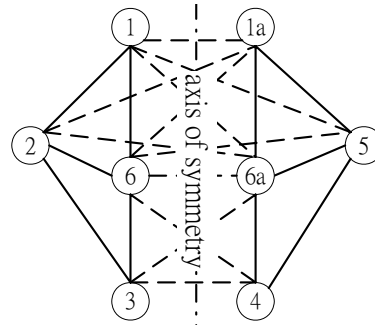
frame. However, this may not always be the case as the number of nodes may be odd or the axis of symmetry may lie on some edges instead of cutting across some edges. The problem can be resolved by splitting of nodes as shown in another example in Figure 36. The split node action is reversible. Hence, the symmetric space frame module can always be formed by re-joining nodes in the resulting adjacency matrix searched by EvoGraph that represents a symmetric space frame module.

In Figure 36(a), the space frame cannot be subdivided into two three-node mirror subframes by deleting edges because the axis of symmetry lies on any one edge connected to node '6'. However, each of the nodes in node set {1, 6} can be split into 2 nodes to give {1, 1a, 6, 6a}. The resulting space frame is shown in Figure 36(b). This extended space frame can be divided into two mirror subframes by deleting edges connecting them (dotted lines in the figure). Subframe {1, 2, 3, 6} is mirror image to subframe {1a, 5, 4, 6a}. This is reflected in the two identical submatrices in the upper left quadrant ( $P$ ) and the lower right quadrant of the adjacency matrix (shaded cells) ( $S$ ) in Figure 36(b), or  $P=S$ . The remaining two subframes in the space frame module connecting subframes {1, 2, 3, 6} and {1a, 5, 4, 6a} are mirror image to each other. This is reflected in the two identical submatrices in the upper right quadrant ( $Q$ ) and lower left quadrant of the adjacency matrix (box cells) ( $R$ ) in Figure 36(b), or  $Q=R$ . As the split node action is reversible, Figure 36(a) can be formed by re-joining the pairs of split nodes, '1' and '1a', '6' and '6a'.



	1	2	3	6	5	4
1	0	1	0	1	1	0
2	1	0	1	1	0	0
3	0	1	0	1	0	1
6	1	1	1	0	1	1
5	1	0	0	1	0	1
4	0	0	1	1	1	0

(a) Pyramid space frame with pentagon base and corresponding adjacency matrix



	1	2	3	6	1a	5	4	6a
1	0	1	0	1	1	1	0	1
2	1	0	1	1	1	0	0	1
3	0	1	0	1	0	0	1	1
6	1	1	1	0	1	1	1	1
1a	1	1	0	1	0	1	0	1
5	1	0	0	1	1	0	1	1
4	0	0	1	1	0	1	0	1
6a	1	1	1	1	1	1	1	0

(b) Node '1' is split into nodes '1', '1a' and node '6' is split into nodes '6', '6a' and corresponding adjacency matrix subdivided into four submatrices of the subframes

**Figure 36:** Example of a Pyramidal Symmetric Space Frame Module with pentagon base divided into subframes which are mirror images to each other

After defining the criteria on determining symmetry of topology of a space frame module, the criteria on symmetry of geometry has to be defined.

### 6.1.2 Symmetric Geometric Properties and Dimensions Design

The Space Frame Module is required to have regular angles between adjacent edges and its edges should have regular lengths to facilitate economy of scale in the repetition process. The regularity of angles and lengths is defined as their capability on being

classified into groups according to their metric values. The smaller the number of groups, the more regular the angles and lengths are. The highest regularity is attained when all the angles and lengths are equal. In this case there is only one group for each of them. Given a topology of space frame module created by EvoGraph, GA should try to find the most regular geometric properties for the final design. Entropy of the geometric properties is introduced to ensure the regularity of geometry. Entropy calculation on angles and lengths are used in this respect.

Let  $|V|$  = number of nodes in a space frame module

$|E|$  = number of edges in a space frame module

$\text{deg}(x_i)$  = degree of node  $i$  in space frame module

$|N_a|$  = number of angles between any two connected edges

$a_j$  = angle  $j$  between two connected edges

$l_k$  = length of edge  $k$

It can be derived that  $\sum_{i=1}^{|V|} \text{deg}(x_i) = 2|E|$  and  $|N_a| = \begin{pmatrix} |E| \\ 2 \\ 2 \end{pmatrix}$

Define

$$\text{Angle Entropy} = H_A = - \sum_{j=1}^{|N_a|} \frac{a_j}{\sum_{j=1}^{|N_a|} a_j} \log_{|N_a|} \left( \frac{a_j}{\sum_{j=1}^{|N_a|} a_j} \right) \quad 0 < H_A \leq 1 \quad (2)$$

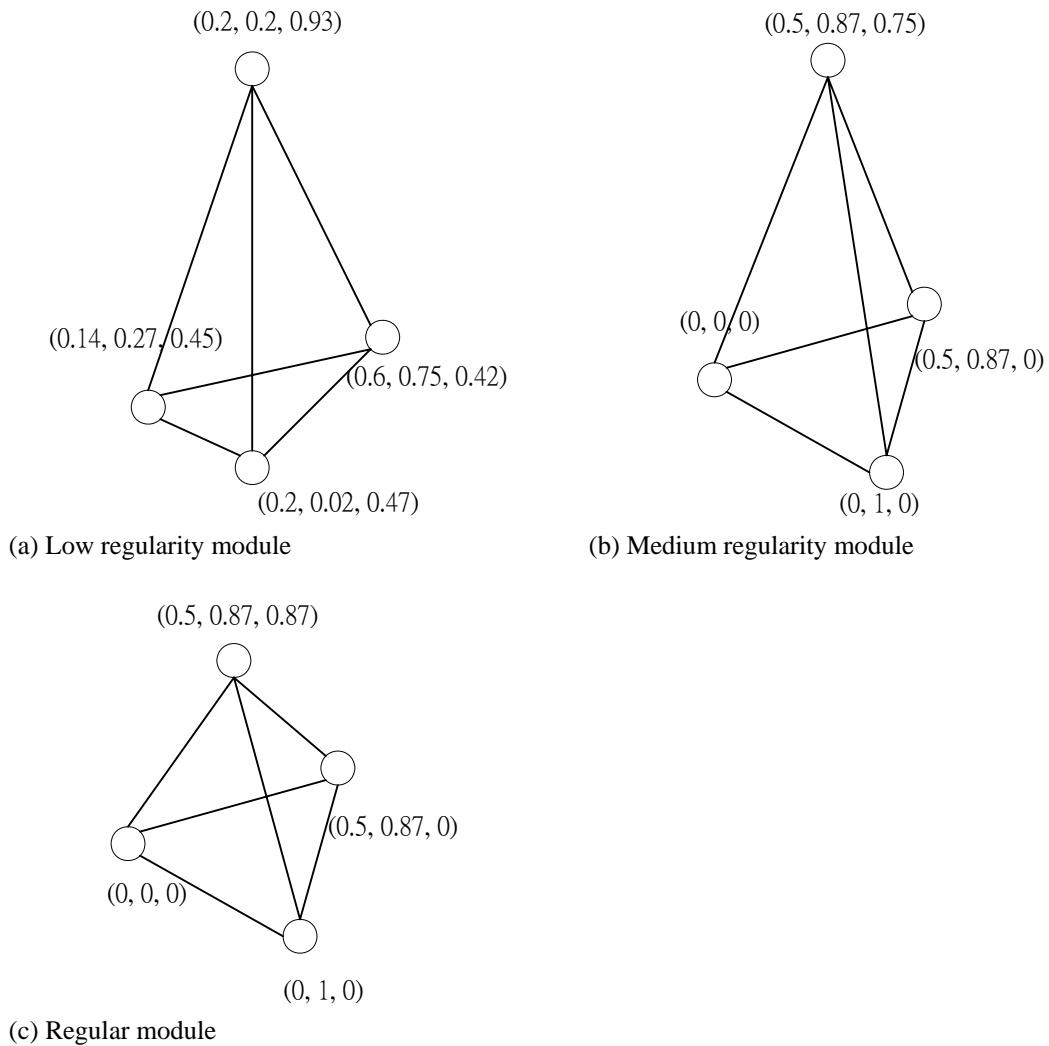
$$\text{Length Entropy} = H_L = - \sum_{k=1}^{|E|} \frac{l_k}{\sum_{k=1}^{|E|} l_k} \log_{|E|} \left( \frac{l_k}{\sum_{k=1}^{|E|} l_k} \right) \quad 0 < H_L \leq 1 \quad (3)$$

Due to the property of the entropy function, the value of  $H_A$  and  $H_L$  increase with the increase in regularity of angles between adjacent edges,  $a_j$ , and length of edges,  $l_k$ , respectively. Hence,  $H_A$  and  $H_L$  are the values to be maximized in the GA fitness function.

If all angles between connected edges are equal, then  $\frac{a_j}{\sum_{j=1}^{|N_a|} a_j} = \frac{1}{|N_a|}$  and  $H_A = 1$

If all edges are equal in length, then  $\frac{l_k}{\sum_{k=1}^{|E|} l_k} = \frac{1}{|E|}$  and  $H_L = 1$ . This is illustrated by an

example below.



**Figure 37:** Examples on tetrahedron with different geometric regularities

Three-dimensional coordinates are assigned to the nodes in Figure 37(a), (b) and (c). Their edge lengths, angles between adjacent edges, length entropies and angle entropies are summarized in Table 7.

	Figure 37(a)	Figure 37(b)	Figure 37(c)
<b>Length Set</b>	0.26, 2x0.50, 0.66, 0.84, 0.85	3x1.00, 3x2.00	6x1.00
<b>Angle Set (radians)</b>	0.27, 0.54, 0.60, 0.62, 0.73, 0.89, 1.2, 3x1.3, 2.1	3x0.50, 9x1.32	12x1.05
$H_L$	0.9642	0.9684	1
$H_A$	0.9524	0.9759	1

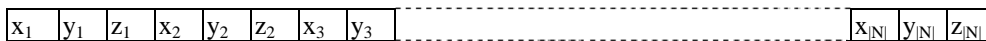
**Table 7:** Regularity of geometric properties in Figures 37(a),(b),(c)

The regularity of edge lengths and angles increases from Figure 37(a) to complete uniformity in Figure 37(c). The trend is reflected in the increase in length entropy,  $H_L$ , and angle entropy,  $H_A$ .

### 6.1.3 Encoding Three Dimensional Coordinates of Space Frame Module Nodes in Linear Chromosome

After obtaining the optimal topology, random three-dimensional coordinates are then assigned to the nodes of the optimal graph to start the GA process to evolve the optimal geometry and dimensions. The three-dimensional coordinates of all the nodes are encoded into a linear chromosome before the GA process. The linear chromosome is shown below.

Let  $x_i, y_i, z_i$  be the three-dimensional coordinates of node  $i$ . All the coordinates are encoded into a linear chromosome in the following form. Without loss of generality, the values of the coordinates are restricted to  $0 \leq x_i, y_i, z_i \leq 1$ .



**Figure 38:** Encoding three dimensional coordinates in linear chromosome

At the end of GA process, the linear chromosomes are subdivided into units with three alleles, each corresponds to a set of three-dimensional coordinates of a node in the space frame module.

## 6.2 Fitness Functions

There are two fitness functions for evaluation on a space frame module. One for the evaluation of the symmetry of the topology and the other evaluates the regularity of geometrical properties and dimensions.

To conduct the fitness evaluation of a space frame module topology, its adjacency matrix should be in the form shown in the examples in Figure 35(b) if the space frame module contains even number of nodes or converted to even number of nodes from odd number by node splitting as shown in the example in Figure 36. Since EvoGraph can evolve the space frame modules using adjacency matrices toward symmetric topology, the evolution of the adjacency matrix to arrive at the four quadrant submatrix form is automatic under proper guidance of the fitness function. Given an adjacency matrix in the four quadrant submatrix form as the example in Figure 35(b), the correlation coefficients between the two pairs of submatrices  $\{A, C\}$  and  $\{B, D\}$  are obtained according to (1). Let them be  $c^{AC}$  and  $c^{BD}$  respectively.

Define the combined correlation coefficient,  $c$ , between the two pairs of submatrices as

$$c = 0.5 c^{AC} + 0.5 c^{BD} \quad \text{where} \quad 0 \leq c, c^{AC}, c^{BD} \leq 1$$



Let  $|E|$  be the number of edges of the space frame module to be evaluated and  $|E_T|$  be the target number of edges of the space frame module

Define fitness function for EvoGraph as follow

$$\text{Maximize } f(|E|, c) = \frac{2}{2^{\|E|-|E_T\|} + 2^{|c-1|}} \quad (4)$$

$\|E|-|E_T\|$  and  $|c-1|$  are the values to be minimized. The integer, 2, is used as the numerator to enable the maximum fitness to reach one when the number of edges of a space frame module equals the target number of edges ( $|E|=|E_T|$ ) and the topology comprises two pairs of mirror subframes ( $c=c^{AC}=c^{BD}=1$ ), or symmetrical topology is achieved. In this case,  $f(|E_T|, 1) = \frac{2}{2^0 + 2^0} = 1$ .

Given the symmetric topology evolved from EvoGraph, the geometrical properties and dimensions of a space frame module are evolved by GA. The fitness function for evaluation of degree of regularity of geometrical properties and dimensions of a space frame module is derived as follow. The objective of GA is to maximize the regularity of geometry and dimensions of the space frame module. But there is a case of regularity when the space frame module collapses. In that case, all the edges are of equal length and the angles between all adjacent edges equal to zero. Hence the angles between the angles and the edges should be maximized to prevent the space frame module from collapsing. To achieve regularity of geometry and dimensions, both  $H_A$  and  $H_L$  should be maximized.

Let  $A$  be the set of angles between all adjacent edges in radians

$L$  be the set of lengths of all edges

$E(A)$  be the mean of all angles between all adjacent edges

Define fitness function for GA as follow.

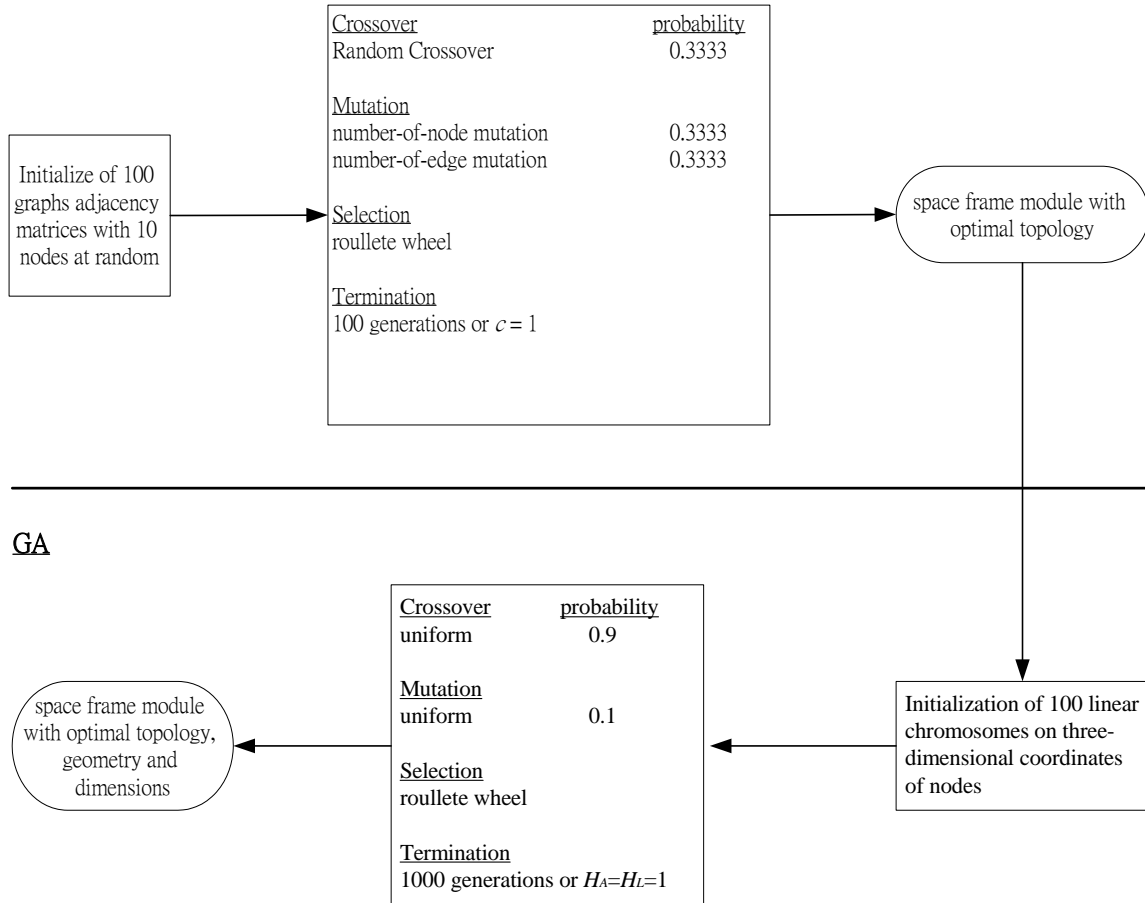
$$\text{Maximize } f(A, L) = E(A) H_A H_L \quad (5)$$

It can be observed that the higher the regularity of the geometrical properties and dimensions, the higher the fitness value. If all the angles between all the adjacent edges and all the length of edges are equal, then  $H_A = H_L = 1$ . The geometrical properties and dimensions of the space frame module have the highest regularity. In this case,  $f(A, L) = E(A)$ .

### 6.3 The Hybrid Evolutionary Algorithm for EvoGraph and GA

In the hybrid evolutionary algorithm, EvoGraph is used to evolve the optimal space frame module topology and GA is used to evolve the optimal geometry and dimensions. The process is summarized in the Figure 39. The parameters used in the process are purported to conduct the experiments on evolution of space frame modules with number of edges between 12 and 20. One hundred graph adjacency matrices are initialized at random for the EvoGraph to evolve the optimal symmetric graph topology. The evolution process follows that described in Section 4.5. There is no bias on the application of the EvoGraph operators. All three basic EvoGraph operators, *random crossover*, *Number-of-Node* mutation and *Number-of-Edge* mutation are applied with equal probability in order to emphasize exploration of search space to exploitation. The termination condition is either reaching the maximum generation of one hundred or the symmetry of topology is reached,  $c=1$ . After the evolution of symmetric topology, the evolution of regular geometric properties and dimensions of the space frame module is evolved by GA. Given the symmetric space frame module topology, one hundred sets of three dimensional coordinates of the nodes are generated at random and each of them are encoded in a linear chromosome as the initial population of GA. Steady state GA using roulette wheel selection is adopted. In order to explore the large search space, uniform crossover and mutation operators are used. The conventional approach of using crossover as the major operator and mutation as the minor operator is adopted. The probability of mutation is 0.1, which is higher than the usually adopted for GA. This is also due to the emphasis on exploration of search space. The termination condition is either the maximum number of generations of 1000 is reached or  $H_A=H_L=1$ .

## EvoGraph



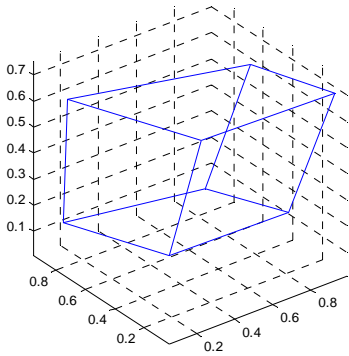
**Figure 39:** Hybrid algorithm process for EvoGraph and GA

## 6.4 Experimental Results

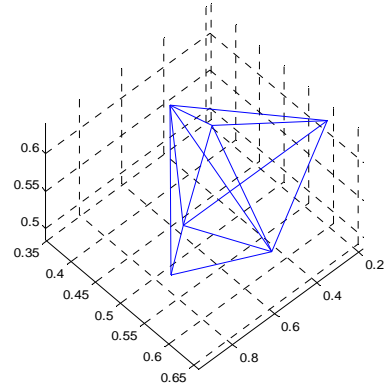
Design parameters with the number of edges ranging from 12 to 20 are input into the hybrid evolutionary algorithm described in Section 6.3 with Steady State Reproduction at each generation. All the topologies are evolved by EvoGraph in not more than 61 generations. The terminal generation on the GA part is fixed at 1000. Near optimal solutions are reached as all the fitness function flattens out near the end of the terminal

generation. The designs evolved give a good regularity of geometry and dimensions as shown in Figure 40. The quantitative results are summarized in Table 8.

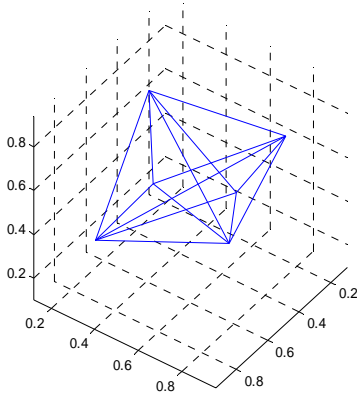
Module A :  $|E| = 12$



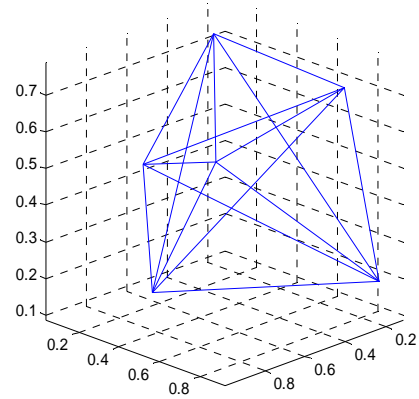
Module B :  $|E| = 13$



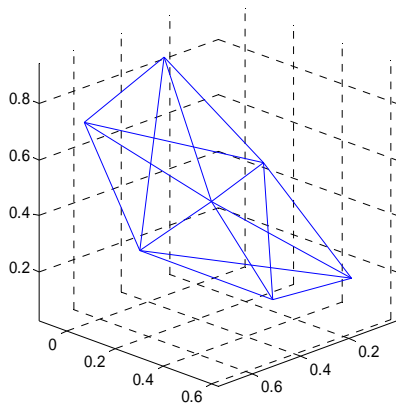
Module C :  $|E| = 14$



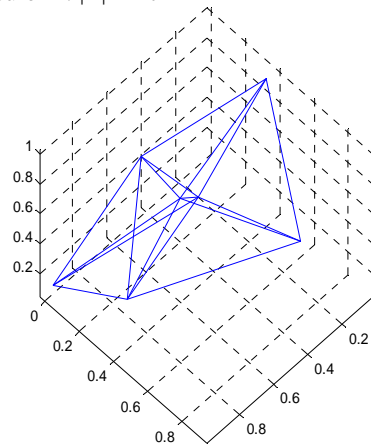
Module D :  $|E| = 15$



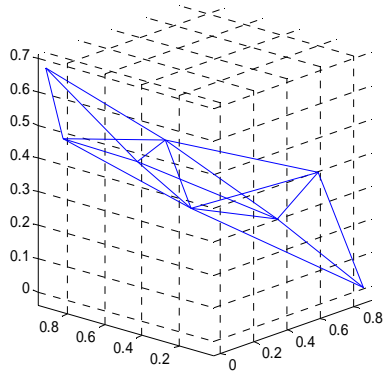
Module E :  $|E| = 16$



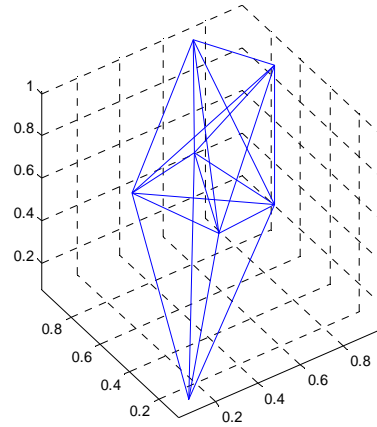
Module F :  $|E| = 17$



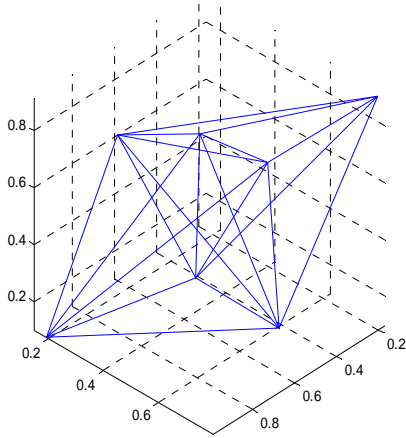
Module G :  $|E| = 18$



Module H :  $|E| = 19$



Module I :  $|E| = 20$

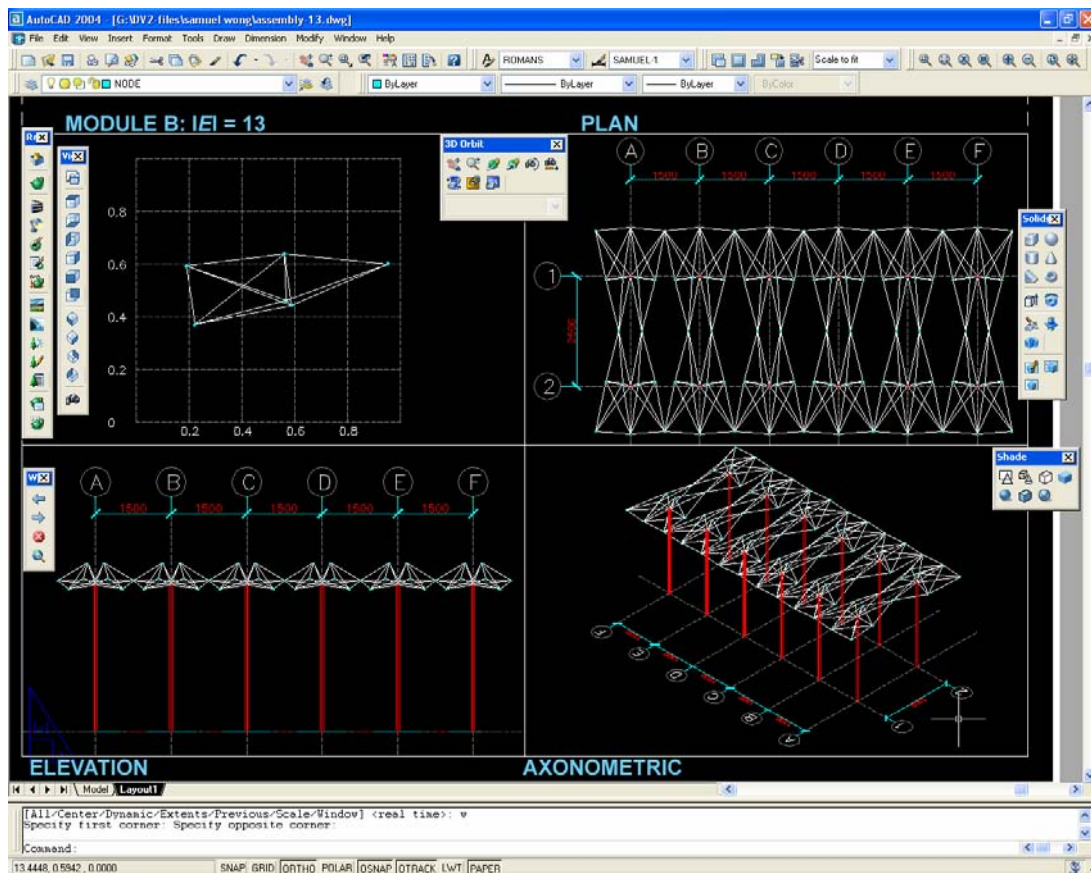


**Figure 40:** Regular space frame modules generated by experiments

No. of edges	No. of generations of EvoGraph to reach optimal topology ( $c = 1$ )	Angle Entropy $H_A$ after 1000 generations of GA	Length Entropy $H_L$ after 1000 generations of GA
12	16	0.9918	0.9952
13	21	0.9661	0.9835
14	6	0.9818	0.9924
15	4	0.9860	0.9915
16	50	0.9600	0.9836
17	9	0.9691	0.9865
18	27	0.9385	0.9733
19	18	0.9760	0.9884
20	61	0.9831	0.9922

**Table 8:** Summary of converging generations and corresponding angle entropies and length entropies of space frame modules

The space frame modules generated can be combined or articulated to form different types of interesting structures depending on their geometric properties and the functions required by a space. Module A in Figure 40 resembles the most popular orthogonal grid space frame structure. This is the most common form of structural framework in design. The other space frame modules have interesting geometries. They are used as basic building blocks of large space of many shapes. Module B to E are diamond shape modules that can be repeated to form building elements such as roof structure. The AutoCAD drawing on repetitive units of module B is shown in Figure 41. Other varieties generated by grouping module C to module I in Figure 40 are illustrated by AutoCAD in Appendix 3.



**Figure 41:** Module B with 13 edges

## Tree Evolution on Art Creation

---

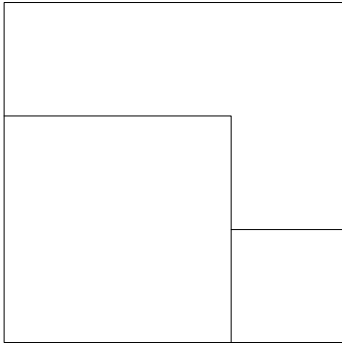
As is pointed out in [55], at the end of the nineteenth century, a tendency within the art movement towards aestheticism took place: the underlying structures and forms of different art forms were seen as the essential basic, most abstract representation. These were based on aesthetic principles that emphasized balance, minimalism and implicit beauty that required mental participation of the viewer to become explicit. The ideas about composition started to become common knowledge. The most basic, generic description of a two-dimensional artwork is called a composition. A composition of a painting can be seen as a formal framework that specifies what a ‘well-formed’ artwork at the highest abstraction-level looks like. Neoplasticism is one of the styles of these modern abstractions. Neoplasticists believed that art should not be the reproduction of real objects, but the expression of the absolutes of life. To them, the only absolutes of life were vertical and horizontal lines and the primary colors - red, yellow and blue (together with white and black).

Piet Mondrian (1872 – 1944) was a Neoplasticist; a devout believer in the ability of art to have a deep spiritual influence on people's lives. He strove throughout his nearly three-decade immersion in pure painting to achieve harmony and balance through an intuitive process of constructing square and rectangular planes of white, red, yellow and blue, dissected by vertical and horizontal black lines. The rules of Mondrian painting are followed and listed below.

1. Only white, red, blue color patches are used.



2. The painting can only contain rectangles with sides parallel to the borders of the canvas. Lines that are not part of the border of a rectangle are not allowed. For example, the following figure is not a proper Mondrian painting line composition. There is a right angle bend that does not form part of a rectangular border.



**Figure 42:** Violation of Mondrian subdivision rule

3. Borders of the rectangle can only be painted in black. Rectangles can only be painted in white, red, blue, or yellow. The borders of the canvas are considered to be black.
4. Two adjacent rectangles cannot have the same color, unless they are painted white.

The drawing process of Mondrian painting encoded in a tree with attributed nodes is shown below. The process is basically a series of rectangular subdivision of the canvas. The edges, terminal nodes and non-terminals have different attributes as listed in Table 9.

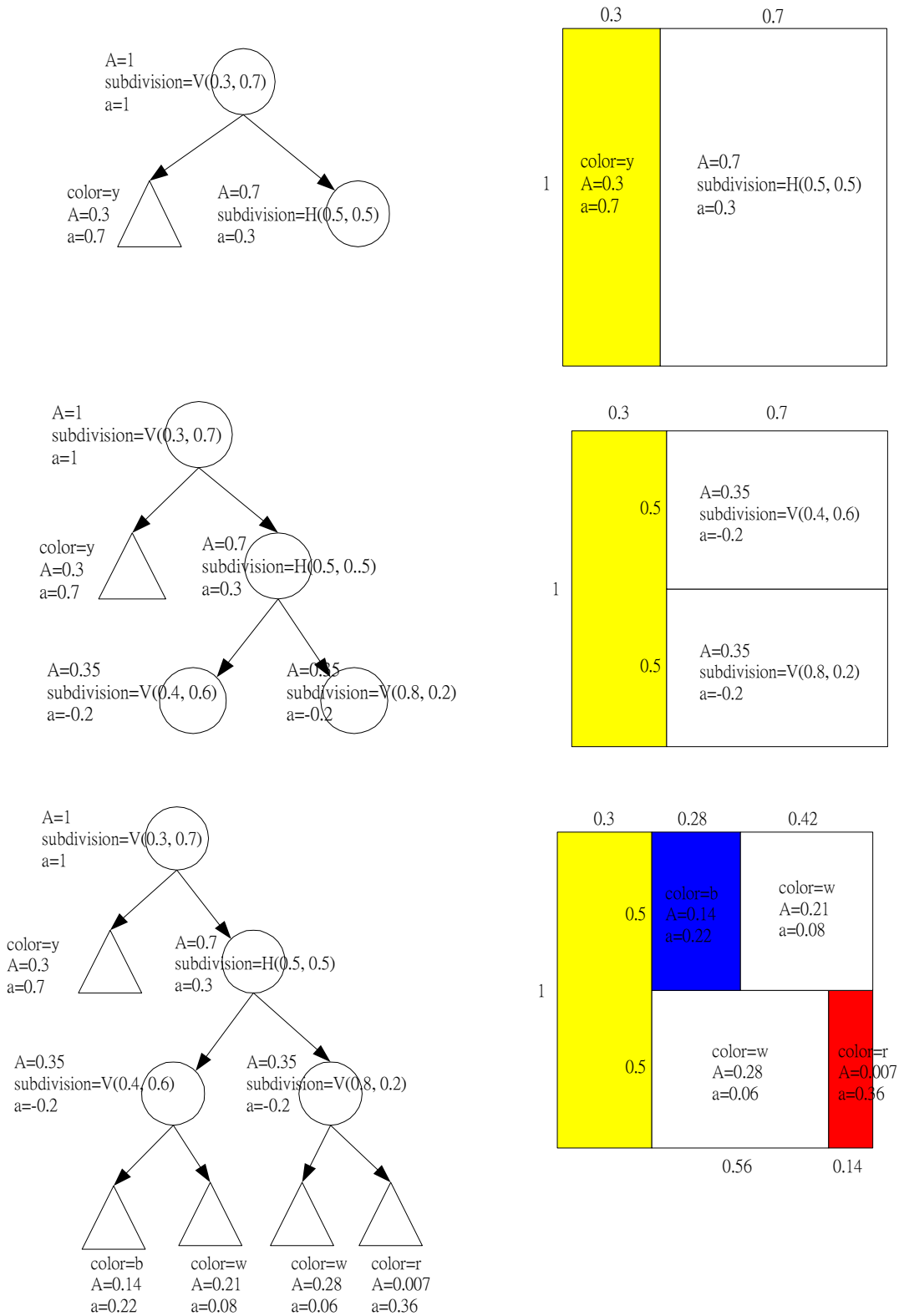
Edge attributes	Terminal node attributes	Non-terminal node attributes
1. area apportionment $x_i$ , $0 < x_i < 1$ and $\sum_{i=1}^k k_i = 1$ where $k$ = no. of subdivisions	1. color: w = white r = red b = blue y = yellow  2. area of subdivided rectangle 3. height-width difference, $a$ , of subdivided rectangle $a = h - w$ where $h$ = height of rectangle $w$ = width of rectangle	1. area of subdivided rectangle  2. direction of subdivision: H = horizontal V = vertical 3. height-width difference, $a$ , of subdivided rectangle

**Table 9:** Symbols for tree encoding of Mondrian painting evolution

An example on evolution of Mondrian painting on a square canvas of unit length is illustrated in Figure 43. The process encoding in the tree is the ‘genotype’ with the drawing expressed as the ‘phenotype’. The subdivisions start from top to bottom and from left to right cascading down the tree. The corresponding subdivisions in the canvas progresses from left to right for vertical subdivision (V) and from bottom to top for horizontal subdivision (H).

Genotype

Phenotype



**Figure 43:** Example on tree encoding of Mondrian painting evolution

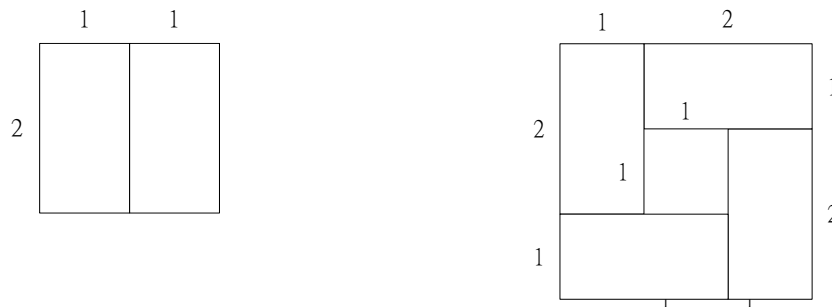
In this process, there are possibilities where two adjacent rectangles having the same color except white color. This contravenes the fourth rule of Mondrian painting described above. The selection process in EvoGraph eliminates this deficiency by assigning lower fitness value to the attributed adjacency matrix that represents the attributed tree of the painting.

## 7.1 Quantification of Aesthetic Attributes of Mondrian Painting

In order to evolve Mondrian painting, several basic aesthetic attributes should be abstracted and quantified. These attributes will be incorporated in the fitness function as the targets of evolution. These aesthetic attributes include the directional bias of the painting, evenness of subdivision, color distribution and granularity of subdivisions.

### 7.1.1 Directional Bias

The sense of direction of the painting is the overall impression on all the directional bias of all the rectangles within the painting. The quantification of this overall sense of direction can be expressed by the weighted height-width difference of the rectangles as illustrated by the example in Figure 44.



(a) Vertical subdivision of a square

(b) Pinwheel subdivision of a square

**Figure 44:** Comparing vertical and pinwheel subdivision of a square on the directional bias

In general, the height-width difference of the terminal subdivided rectangles is used to determine the overall directional bias of the painting, as they are the most conceivable. In the above figures, there are two rectangles in Figure 44(a) and four rectangles and one square in Figure 44(b). The overall visual impact of each subdivided rectangle is proportional to its area. Hence their relative areas weight the overall height-width difference of the figures.

$$\text{weighted height-width difference } a = \sum_{i=1}^n \frac{A_i}{A} a_i$$

where  $a_i$  = height-width difference of rectangle  $i$  in the painting

$A$  = area of canvas

$A_i$  = area of the  $i$ th rectangle in the painting

$n$  = total number of subdivided rectangles

$$\sum_{i=1}^n A_i = A \quad \text{hence, total area considered including the canvas} = 2A$$

if  $a < 0$ , the figure is horizontally biased

if  $a = 0$ , the figure is unbiased

if  $a > 0$ , the figure is vertically biased

Figure	$A$	$a_i$	$A_i/A$	$a$
44(a)	4	$2 - 1 = 1$ $2 - 1 = 1$	$2/4$ $2/4$	$1 \times \frac{2}{4} + 1 \times \frac{2}{4} = 1 > 0$
44(b)	9	$2 - 1 = 1$ $1 - 2 = -1$ $2 - 1 = 1$ $1 - 2 = -1$ $1 - 1 = 0$	$2/9$ $2/9$ $2/9$ $2/9$ $1/9$	$1 \times \frac{2}{9} - 1 \times \frac{2}{9} + 1 \times \frac{2}{9} - 1 \times \frac{2}{9} + 0 \times \frac{1}{9} = 0$

**Table 10:** Calculation of degree of bias of Figures 44(a) and (b)

Figures 44(a) is biased vertically ( $a = 1 > 0$ ) and there is no directional bias in Figure 44(b) as all the directions of the subdivision rectangles balance out each other ( $a = 0$ ). The square in the centre is unbiased. This interpretation tallies with our visual perception.

### 7.1.2 Evenness of Subdivision

The entropy of all rectangular areas within the painting measures the evenness of area of subdivision in a Mondrian painting.

Let  $A$  = area of canvas

$A_i$  = area of the  $i$ th rectangle in the painting

entropy of areas  $H_n = -\sum_{i=1}^n \frac{A_i}{A} \log_n \frac{A_i}{A}$  where  $\sum_{i=1}^n \frac{A_i}{A} = 1$

The higher the value of  $H_n$ , the more uneven the areas are distributed. When the areas are evenly distributed,  $H_n = 1$ .

### 7.1.3 Color Distribution

The color distribution within the painting is described by the tuple

$(p_w, p_r, p_b, p_y)$  where

$p_w$  = proportion of area of all white color rectangles in the painting

$p_r$  = proportion of area of all red color rectangles in the painting

$p_b$  = proportion of area of all blue color rectangles in the painting

$p_y$  = proportion of area of all yellow color rectangles in the painting

and  $p_w + p_r + p_b + p_y = 1$

### 7.1.4 Granularity of Subdivision

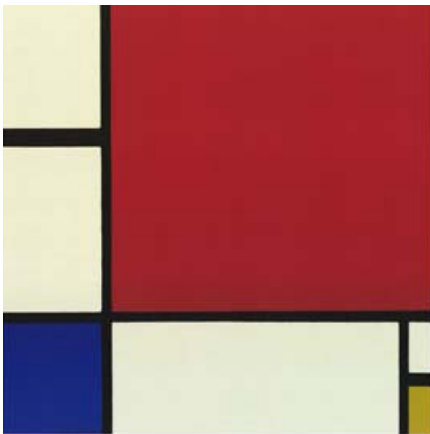
The granularity of subdivision is proportional to the number of subdivided rectangles,  $n$ , in the painting. The attribute values of the painting in Figure 43 are summarized as follow.

Attributes	Calculations
Directional Bias	$A = 1, n = 5$ $0.3 \times (1 - 0.3) + 0.14 \times (0.5 - 0.28) + 0.21 \times (0.5 - 0.42)$ $+ 0.28 \times (0.5 - 0.56) + 0.07 \times (0.5 - 0.14) = 0.266 > 0$
Evenness of Subdivision	$H_a = -0.3 \log_5 0.3 - 0.14 \log_5 0.14 - 0.21 \log_5 0.21$ $- 0.28 \log_5 0.28 - 0.07 \log_5 0.07 = 0.94$
Color Distribution	$p_w = 0.21 + 0.28 = 0.49,$ $p_r = 0.07,$ $p_b = 0.14,$ $p_v = 0.3$
Granularity of Subdivision	$n = 5$

**Table 11:** Attribute values calculation for Mondrian Painting in Figure 43

## 7.2 Experiments

An original painting by Mondrian, ‘Composition with Red, Blue, Yellow’, is selected for the experiment. The attributes of this painting are quantified and extracted. EvoGraph is used to create new paintings by manipulating the values of the extracted figures. The new paintings created are expected to preserve some but not too many of the attributes of the original painting to avoid duplicating the original painting.



**Figure 45:** Original Mondrian Painting ‘Composition with Red, Blue, Yellow’

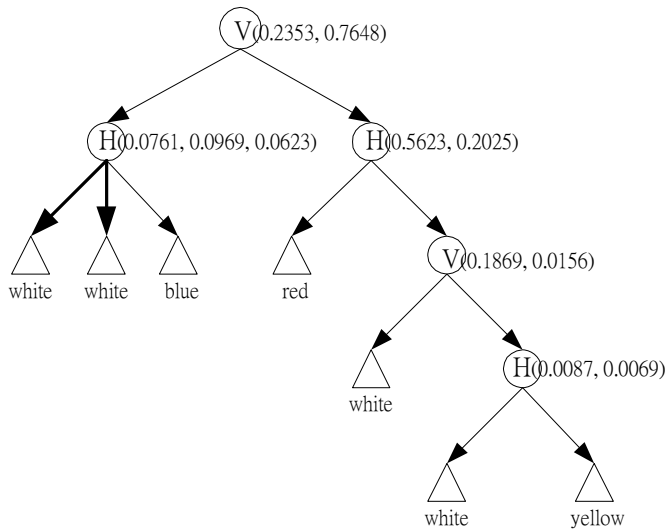
The size of the original painting is normalized to a square of unit area for the ease of calculation. The attributes extracted are the color Area, the directional bias, evenness of area subdivision and granularity of subdivisions. Their respective values are summarized as follow.



Color Area				Directional Bias of Subdivided Rectangles	Evenness of Subdivided Area	Granularity of Subdivision
White	Red	Blue	Yellow	Weighted height-width difference	Entropy of Areas of Rectangular Subdivisions	Number of Rectangular Subdivisions
0.3685	0.5623	0.0623	0.0069	0.1258	0.6727	7

**Table 12:** Attribute values for ‘Composition with Red, Blue, Yellow’

The original painting can be expressed in form of attributed tree and attributed adjacency matrix as follow. For simplicity of presentation, the directions of subdivisions and the area of subdivisions are labeled in the circular non-terminal nodes. The colors of the subdivisions are labeled on the triangular terminal nodes. Thick black lines occasionally exist in some of the Mondrian paintings. The same exists in the left part of the above original painting. The thick arrows in the tree indicate the thicker line partition between the two white rectangles. This can be created by random assignment.



**Figure 46(a):** Tree encoding on evolution of ‘Composition with Red, Blue, Yellow’

	V	H	H	w	w	b	r	V	w	H	w	y
V	0	0.2353	0.7648	0	0	0	0	0	0	0	0	0
H	0	0	0	0.0761	0.0969	0.0623	0	0	0	0	0	0
H	0	0	0	0	0	0	0.5623	0.2025	0	0	0	0
w	0	0	0	0	0	0	0	0	0	0	0	0
w	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0	0
r	0	0	0	0	0	0	0	0	0	0	0	0
V	0	0	0	0	0	0	0	0	0.1869	0.0156	0	0
H	0	0	0	0	0	0	0	0	0	0	0.0087	0.0069
w	0	0	0	0	0	0	0	0	0	0	0	0
y	0	0	0	0	0	0	0	0	0	0	0	0

non-terminal nodes: V = vertical subdivision H = horizontal subdivision

terminal nodes: w = white r = red b =blue y = yellow

**Figure 46(b):** Attributed adjacency matrix for the tree in Figure 46(a)

### 7.2.1 Fitness function

Given the values of attributes extracted from the original Mondrian painting,  $g_i$ , where  $1 \leq i \leq n$ , the general form of fitness function to be maximized for a new painting evolved is

$$\text{fitness} = \frac{1+n}{2^{\text{adj}} + \sum_{i=1}^n 2^{|c_i - g_i|}}$$

$n$  = number of extracted attributes values

$\text{adj}$  = number of pair of adjacent rectangles that are not white and have the same color

$g_i$  = value of attributes  $i$  of original painting

$c_i$  = value of attributes  $i$  of new painting

The value on violation of Mondrian painting rule and deviation from the aesthetic attributes in the denominator are the values to be minimized. Power function is used for the denominator to avoid its value to become zero. '1+n' is used as numerator to enable the fitness to become one at the highest fitness where  $\text{adj} = 0$  and  $|c_i - g_i| = 0$  for all  $i$ . At the highest fitness value, all the constraints are being satisfied. Hence,  $0 < \text{fitness} \leq 1$ .

## 7.2.2 Experiment Setup

The effect of mixing GP crossover and EvoGraph *random crossover* of trees is tested in the experiments. A mix of GP crossover and EvoGraph *random crossover* is used in each generation of evolution. The total number of GP crossover and EvoGraph *random crossover* in each mix is 10. There are total 11 GP crossover / EvoGraph *random crossover* proportional mixes in each experiment. The first mix starts off with pure EvoGraph (10 numbers EvoGraph and no GP). One EvoGraph *random crossover* is replaced by GP crossover in the second mix and so on. At the 11<sup>th</sup> mix, there is pure GP (10 numbers GP crossover and no EvoGraph *random crossover*). The arrangement is summarized as follow.

Mix	No. of GP crossover	No. of EvoGraph <i>random crossover</i>
1	0	10
2	1	9
3	2	8
4	3	7
5	4	6
6	5	5
7	6	4
8	7	3
9	8	2
10	9	1
11	10	0

**Table 13:** Design mix of GP and EvoGraph for Mondrian painting evolution

Five sets of experiments are set up. In each set of experiments, all the 11 mixes in Table 13 are run. The 7 attribute values of the Mondrian painting in Figure 45 are used as search targets of the experiments. In each experiment set, a specific number of targets are to be satisfied by the evolutionary algorithm as indicated in Table 14. Experiment sets 1 and 2 both retain the number of subdivisions with the former capturing aspect ratios and

the latter captures the area distribution of the rectangles. Experiment set 3 captures the aspect ratios, number of subdivisions and the area distribution of the rectangles. Experiment set 4 captures all the color proportions and the area distribution of the rectangles. Experiment set 5 captures all the attributes from the original painting. A resulting painting of very similar nature to the original is expected.

Sets of Experiment	Attributes and Target Values						
	white	red	blue	yellow	Weighed height-width difference	Area Entropy	no. of subdivisions
	0.3685	0.5623	0.0623	0.0069	0.1258	0.6727	7
1	-	-	-	-	Y	-	Y
2	-	-	-	-	-	Y	Y
3	-	-	-	-	Y	Y	Y
4	Y	Y	Y	Y	-	Y	-
5	Y	Y	Y	Y	Y	Y	Y

**Table 14:** Matrix identifying attributes to be captured in different experiments

The problem complexity increases with the number of target attribute values to be matched and the nature of the complexity of attribute value derivation. The easiest to match target is the number of subdivisions which is an integer. The next more complex targets are the color proportions which are decimals. The most difficult are the aspect ratio and area entropy which require mathematical derivations. Experiment sets 1 and 2 will have almost the same complexity. The complexity increases from experiment set 3 to experiment set 5 as the number of target attribute value increases.

The fitness value at convergence and the number of generations to reach convergence with respect to different mixes of GP crossover and EvoGraph *random crossover* in each experiment are studied. Furthermore, the cross sectional analysis on general convergence pattern amongst all the experiments will also be conducted.

### 7.2.3 Evolution Parameters and the Evolution Process

An initial population of 50 trees is generated at random. All experiments are based the same initial population so that comparison of performance can be made. Convergence is considered to be reached if the fitness value equals or exceed 0.99 which is close enough to 1 or there is no further increase in fitness value for 1000 consecutive generations. Steady state reproduction is used in the evolution process. The maximum fitness of a generation is a monotonic increasing function of the number of generations of evolution. The converging generation is the first generation that reaches or exceeds fitness value of 0.99 or the beginning generation of the 1000 consecutive stagnant fitness. Both standard GP crossover and the EvoGraph *random crossover* operation for trees described in sections 4.2.2 and 4.2.3 are used. The crossover operators evolve the tree topology of the genotype. The attributes of the tree (subdivision proportion, direction of subdivision, color) are searched by mutations of values in the attributed adjacency matrix. Each crossover is followed by one mutation of attribute values. Steady State Reproduction is adopted. The evolutionary process is listed as follow.

1. Generate an initial population of attributed adjacency matrices randomly.
2. Evaluate each attributed adjacency matrix according to their fitness.
3. According to respective numbers of GP crossover and EvoGraph *random crossover* in Table 13, for each cross over operation select two attributed adjacency matrices with attribute values for reproduction using the Roulette Wheel selection scheme.
4. Carry out crossover and mutation and reproduce.
5. Evaluate fitness of the resulting children adjacency matrices with attributed values.

6. Delete two least-fit individuals from current population and insert new adjacency matrices (Steady State Reproduction).

7. Repeat from 3 until termination criteria are reached.

#### 7.2.4 Fitness Analysis

The converging fitness values of the experiments are summarized in Table 15. Least square polynomial regression analysis is conducted for the normalized fitness gap for each experiment. The regression error is measured by the  $R^2$  value defined below.  $0 \leq R^2 \leq 1$ , when  $R^2 = 1$ , the curve is a perfect fit to the input values and vice versa.

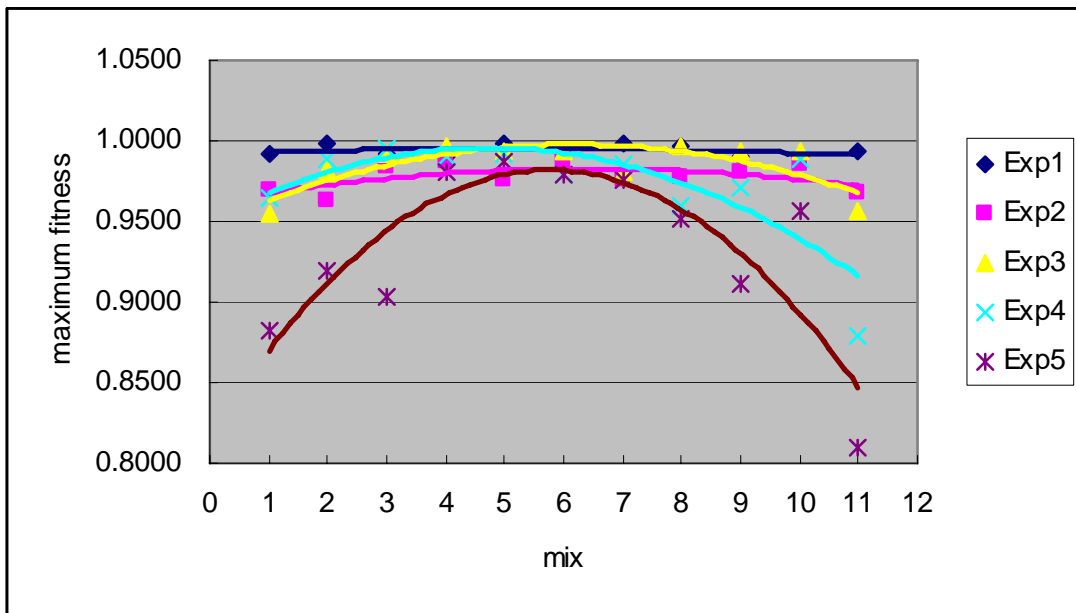
$$R^2 = 1 - \frac{\sum_{i=1}^{11} (Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{11} Y_i^2 - \frac{(\sum_{i=1}^{11} Y_i)^2}{11}}$$

where  $Y_i$  = actual value at  $i$   $\hat{Y}_i$  = expected value on regression line at  $i$

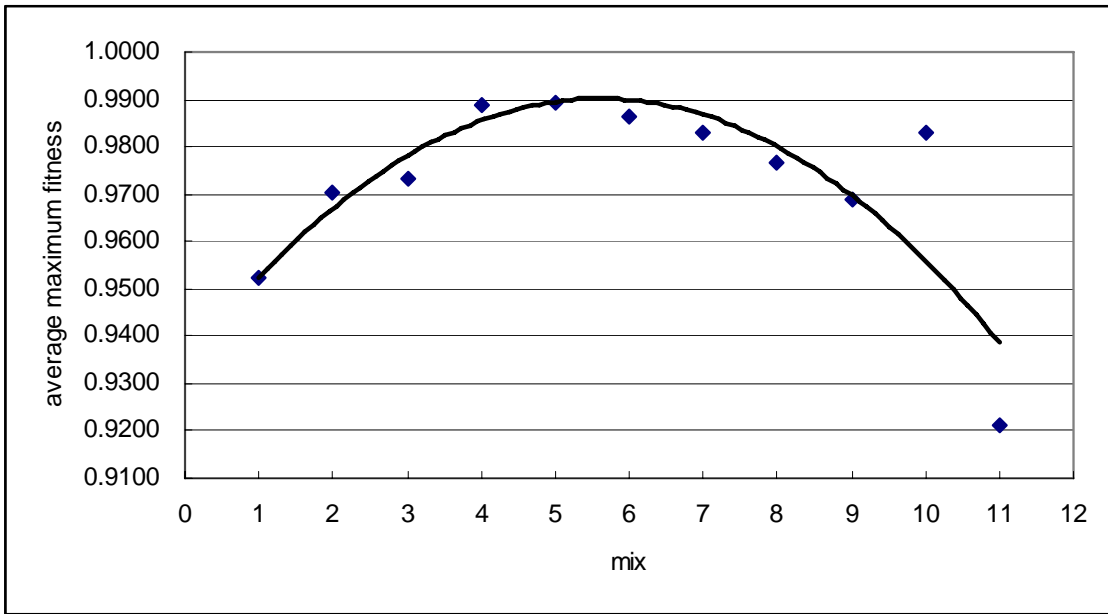
The regression curve of each experiment w.r.t. the crossover mix is plotted in Figure 47. The regression curve of the average fitness of the 5 experiments w.r.t. the crossover mix is plotted in Figure 48. The regression curve of average maximum fitness of the 11 crossover mixes w.r.t. the 5 experiments in the order of increasing complexity of search is plotted in Figure 49. The paintings evolved by the best fit mixes in the five experiments are illustrated in Appendix 4.

mix	Exp Set 1	Exp Set 2	Exp Set 3	Exp Set 4	Exp Set 5	average
1	0.9914	0.9693	0.9551	0.9645	0.8818	<b>0.9524</b>
2	0.9988	0.9635	0.9817	0.9884	0.9194	<b>0.9704</b>
3	0.9924	0.9838	0.9922	0.9956	0.9035	<b>0.9735</b>
4	0.9907	0.9877	0.9960	0.9909	0.9800	<b>0.9890</b>
5	0.9989	0.9751	0.9940	0.9925	0.9864	<b>0.9894</b>
6	0.9924	0.9878	0.9932	0.9783	0.9797	<b>0.9863</b>
7	0.9982	0.9754	0.9802	0.9856	0.9756	<b>0.9830</b>
8	0.9962	0.9770	0.9973	0.9599	0.9522	<b>0.9765</b>
9	0.9905	0.9800	0.9930	0.9710	0.9108	<b>0.9691</b>
10	0.9901	0.9858	0.9933	0.9887	0.9558	<b>0.9827</b>
11	0.9932	0.9674	0.9571	0.8789	0.8098	<b>0.9213</b>
average	<b>0.9939</b>	<b>0.9775</b>	<b>0.9848</b>	<b>0.9722</b>	<b>0.9323</b>	

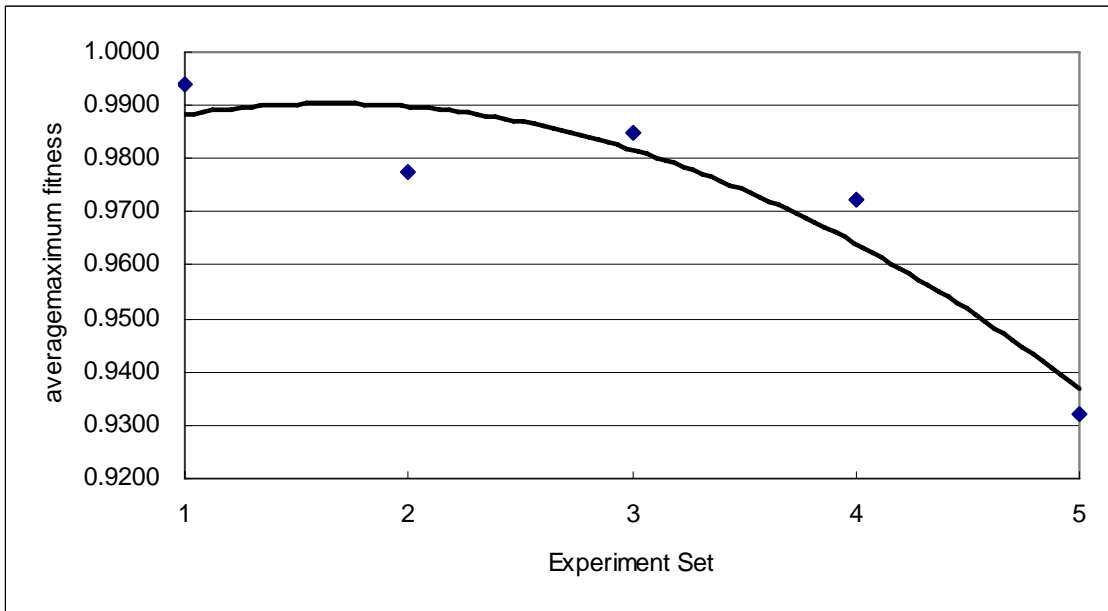
**Table 15:** Converging fitness value



**Figure 47:** Regression curve of maximum fitness of experiment sets w.r.t. crossover mix



**Figure 48:** Regression curve of average maximum fitness of the 5 experiment sets w.r.t. crossover mix



**Figure 49:** Regression curve of average maximum fitness of the 11 crossover mixes w.r.t. the 5 experiment sets in the order of increasing complexity of search



The following can be observed on fitnesses of the experiments.

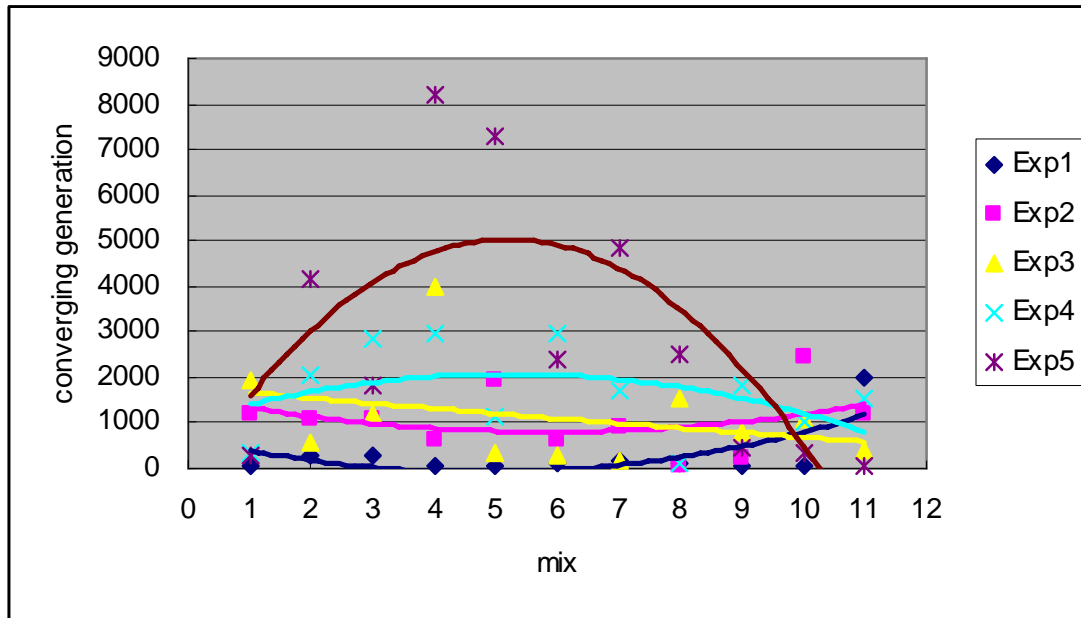
1. There is no significant difference in the maximum fitness evolved for low complexity search from experiment set 1 to experiment 3 as indicated in Figure 47. As the complexity of search increases in experiments set 4 and 5, there is significant drop in maximum fitness at both ends of the curve. The left end of the curve (with higher proportion of EvoGraph crossover) has maximum fitness higher than the right end (with higher proportion of GP crossover).
2. Except the anomaly in mix 10 in Figure 48, there is a tendency of highest maximum fitness to be evolved around mix 5 and 6 where the relative proportion of EvoGraph *random crossover* and standard GP crossover are almost equal.
3. The average maximum fitness evolved by pure EvoGraph *random crossover* (mix 1) is higher than that of pure standard GP crossover (mix 11) as shown in Figure 48.
4. The maximum average fitness of the experiment set of the 11 mixes decreases with increasing complexity of search as indicated Figure 49.

### 7.2.5 Converging Generation Analysis

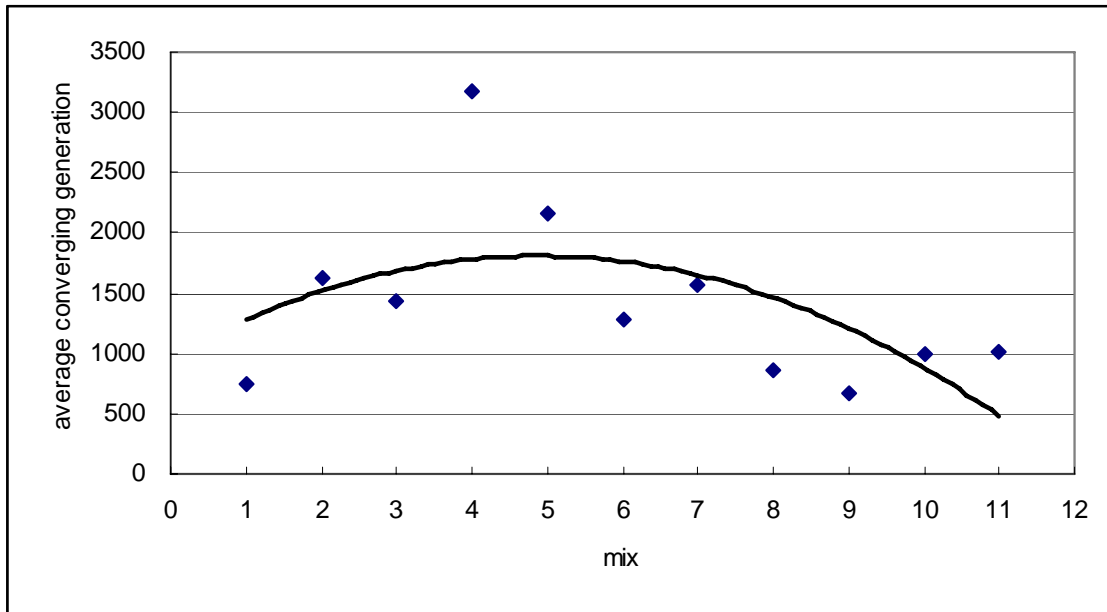
The number of generations to reach convergence (converging generation) is summarized in Table 16.

mix	Exp Set 1	Exp Set 2	Exp Set 3	Exp Set 4	Exp Set 5	average
1	44	1197	1908	342	274	753
2	278	1083	542	2042	4170	1623
3	261	1061	1183	2835	1846	1437
4	69	603	4003	2982	8229	3177
5	70	1931	331	1139	7298	2154
6	87	649	297	2946	2400	1276
7	166	907	174	1686	4867	1560
8	90	81	1536	130	2487	865
9	84	212	799	1827	436	672
10	78	2432	1072	1010	337	986
11	1978	1173	372	1517	48	1018
average	291	1030	1111	1678	2945	

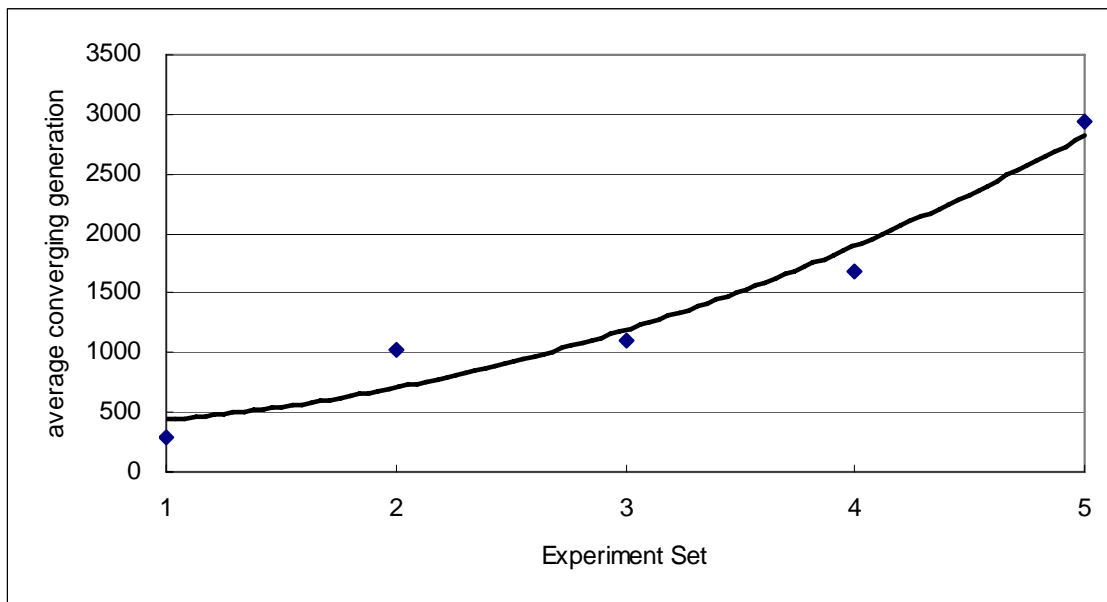
**Table 16:** Number of generations to reach convergence



**Figure 50:** Regression curve of the converging generation of experiment sets w.r.t. crossover mix



**Figure 51:** Regression curve of average converging generation of the 5 experiment sets w.r.t. crossover mix



**Figure 52:** Regression curve of average converging generation of the 11 crossover mixes w.r.t. the 5 experiment sets in the order of increasing complexity of search

The following can be observed from the converging generation of the experiments.

1. According to Figure 50, the difference in converging generation number from experiment set 1 to 4 across all the 11 crossover mixes is not significant. There is significant increase in the converging generation on experiment set 5 which complexity of search is the highest among all the experiments. The regression curves change from concave in experiment 1 gradually to convex in experiment set 4 and 5. This indicates the converging generation increase significantly especially between mix 5 and 6 where the proportion of EvoGraph *random crossover* and standard GP crossover are almost equal. This region of mix is also where the maximum fitness of all experiments are the highest as discussed in the fitness analysis in Section 7.2.4
2. Except the anomaly of mix 4 in Figure 51, there is a trend on having the highest average converging generation for all the experiments between mix 5 and 6. The trend is weaker than its counterpart in fitness analysis having highest maximum fitness in the same region as the data points are scattered further away from the regression curve in the figure.
3. The average converging generation evolved by pure EvoGraph *random crossover* (mix 1) is lower than that of pure standard GP crossover (mix 11) as shown in Figure 51 but it is close to that of mix 8 and 9 as shown in the same figure.
4. The converging generation increases with the complexity of search from experiment set 1 to experiment set 5 as shown in Figure 52.

Referring to the fitness analysis and converging generation analysis, it can be concluded

that though EvoGraph *random crossover* attains higher maximum fitness value and faster convergence in comparing with standard GP crossover, it has the same tendency as the latter on early convergence. The fitness can be improved by mixing it with standard GP crossover at nearly the same proportion but the drawback is that it takes longer time to converge.

---

## Molecule Design

---

The molecular design problem is concerned with the determination of a molecular structure with certain desirable properties. An effective solution to this problem can have many applications in many areas in the biochemical and pharmaceutical industry. Traditionally, molecular design is by many trials-and-errors in laboratories and is difficult, time consuming and expensive. To facilitate the design process, various computer-aided molecular design (CAMD) techniques have been developed and they have been divided into five categories [101]: random search, heuristic enumeration, mathematical programming, knowledge-based systems and graphical reconstruction methods. The various drawbacks, including combinatorial complexity of the search space and the non-linear structure-property correlations, etc., that these techniques have are discussed in [101].

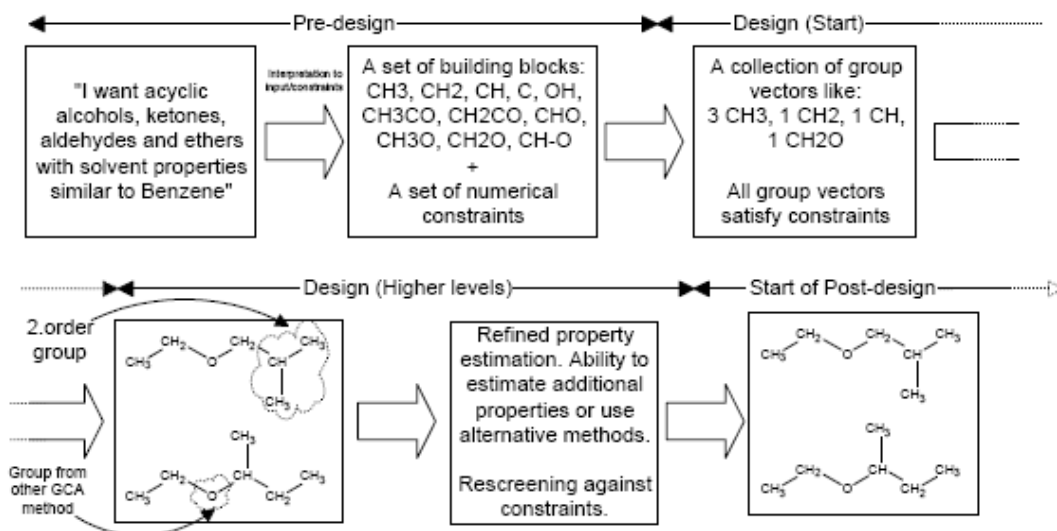
To overcome these drawbacks, there have been some attempts to use EAs to evolve molecular designs encoded in linear strings of parameters [101], trees [102] or genetic graphs [18]. There have also been attempts to use a hybrid of Back Propagation Neural Networks (BPNN) and GAs to evolve molecular designs encoded as graphs [103][104]. For these EAs to work effectively, components in the target molecule and their chemical combination rules have to be known in advance and used from the beginning to guide the evolutionary process [101][102][18]. The chemical combination rules are needed to restrict the variety of molecular structures that can be generated during the evolutionary process to ensure convergence of the process in reasonable time. The chemical

combination rules are implemented in the crossover operators so that when molecular components are exchanged, the crossover operations follow certain orders defined by the rules.

The requirement for molecular components and chemical combination rules to be known ahead of time restricts, to some extent, the ability of existing EA based CAMD technique to evolve novel molecular structures. To allow for greater flexibility, EvoGraph performs its tasks by encoding molecular structures in molecular graphs with nodes in these graphs representing atoms and edges representing bonds. Each molecular graph is in turn encoded into adjacency matrices that represent nodes and their connectivity at the same time. Given such an encoding scheme, the EvoGraph is able to evolve an “optimal” molecular design using a set of reproduction operators designed specifically to handle molecular graphs.

### **8.1 Existing Approaches for Computer Aided Molecular Design**

The traditional approaches to molecular design require many laborious iterations of design-synthesis-evaluation and they are expensive and time consuming. To speed up the design process, various computer-aided molecular design (CAMD) techniques have been developed. CAMD can be defined as follows: “Given a set of building blocks and a specified set of target properties, determine the molecule or molecular structure that matches these properties” [105]. The basic steps of CAMD, according to this definition can be divided into three phases: a pre-design, a design and a post-design stage as shown in Figure 53.



**Figure 53:** Basic steps of CAMD [105]

In the pre-design phase, it should be noted that building blocks of molecular components have to be identified first in order for the target molecules to be formed. For the rest of the other design phases, computer aided design software are often used to order and combine different building blocks to form new designs. Mathematical programming or a hybrid of mathematical and qualitative approaches [106] have also been used to simultaneously design the structure of new molecules and test its optimality using mathematical formulae that model the properties of the target molecule. The major drawbacks of these approaches are due to the need to deal with the combinatorial complexity of the search space and the difficulties in tackling non-linear structure-property correlations, etc. [101].

Recently, there have been attempts to use EAs to generate molecular structures [107]. Given selected building blocks, EAs have been used to evolve the topology of a molecule by linking building blocks to form molecules. If the topology of a molecule is given, EAs can also be used to evolve the relative position of elements and internal bond angles



within the molecule. These EAs adopt different encoding schemes and these schemes impose different constraints on the EA operations that have to mimic chemical reactions when components and bonds between molecules are exchanged. In [24], for example, molecules are encoded in linear strings which represent elemental, substructural, or monomer units. Genetic Algorithms (GAs) are used with these linear strings to evolve target molecules. In [103][104], a GA is also used to search for combinations of molecular fragments encoded as linear strings. The fitness of the candidate molecules are evaluated using back-propagation neural network (BPNN). In [18], molecular design is encoded first as trees for a GP based algorithm. Edges are added at random whenever needed to form rings to create *genetic graphs*.

In addition to the above, EA based CAMD methods have been used to tackle two molecular design problems involving protein ligand docking and variable selection for the development of quantitative structure-activity relations (QSAR) to generate molecular structure with the selected variables.

Protein ligand docking [60] refers to the non-covalent binding of a molecule to a protein. It is widely used in drug design in which the molecular structure of a drug is designed to tackle a particular protein. DOCK [109], for example, is a computer-aided drug molecule design software developed for this purpose. It provides a software interface to facilitate manual manipulation of various molecular structures. With such a feature, DOCK can help users to more easily find the molecular structure of an inhibitor as a docking ligand on a macromolecule receptor. This resulting ligand structure is then matched against a database of computer-derived structures of putative ligands using isomorphic subgraph matching. Although complete matching of two molecular

structures is not always possible, it does provide rapid heuristic approximations and it allows thousands of alternative geometric matches to be examined per second. A variation on the DOCK approach is adopted by an EP-based program called EPDOCK [108]. EPDOCK finds optimal internal bond angles and position of the atoms in the ligand to bind to the target sites of another protein molecule [108]. To do so, it requires that the topology of a ligand be given.

QSAR is a mathematical description of a molecule's physical or chemical properties. It has been shown that EAs can be used to generate target molecules using QSAR properties as search criteria. The evolutionary processes require that known molecular components be provided from the beginning. The target molecule is then evolved by combining individual components in different molecules or through attachment of new components to existing ones to form the target molecule. With the molecular topology given and known molecular components connected with right adjacencies, a microscopic search can then be conducted using EA to confirm that the components within the molecule are arranged in optimal relations with each other.

From a given base molecule, EAs can therefore be used to vary a molecule's bond angles, inter-atomic distances, and atomic forces to achieve minimum energy state in which the molecule is most stable [62][110]. This is known as conformational search as the search is constrained only to molecules conforming to specific molecular structures. Many EA based CAMD on conformational search has been reported [61][111]-[112],[114]-[115],[117]-[120]. They also make use of such constraints as nuclear magnetic resonance spectrometry, as determined from experiments, as a criterion to guide the search [18]-[19]. The use of GA, compared with other parallel direct search methods,

have been studied and reported in [110]. Other than these EA based approaches, methods such as TORK [113] can also be considered as a conformational search algorithm. TORK was developed to adjust internal bond angles of a molecule to achieve minimum energy state. For these conformational search algorithms to work effectively, it should be noted the structure of the target molecules have to be known.

While most of these EA based approaches are based on conformational search, there is one exception -- the *de novo design* approach [59][116]. In *de novo design*, a molecule is designed without reference to any known molecule. It starts off with some potential components of the target molecule and evolves according to principles for linking individual components to form a target molecule. Even though this design approach does not require as much domain knowledge as with other conformational search approaches, potential components that are likely to produce the target and the corresponding combining rules of these components have to be known and determined in advance.

In summary, the current EA approaches to molecular design are either conformational search approaches requiring close-to-target molecular structures or molecular components of the target and their corresponding chemical combination rules be made known. The optimal target molecules that may be discovered may be constrained by the structure of the components that are available.

To be able to design target molecule without all these prior information about molecular structures, molecular components or the chemical combination rules, EvoGraph algorithm is proposed. The algorithm begins with an initial population of random combinations of atoms and simple elements, such as carbon and benzene rings without the need for sophisticated potential building blocks such as phenol groups,

methyl groups etc., to be given ahead of time. Also, the evolutionary process that EvoGraph goes through is guided by molecule descriptors composed of topological indices that describe the adjacency of elements within the target molecule. The advantage of EvoGraph is that it can design molecules without the need for the knowledge of close to target molecular components and the chemical combination process required to reach the target.

## 8.2 Encoding Molecular Design

Existing EA based CAMD methods encode molecular designs in different schemes. For example, linear-strings have been used in [101] to encode polymers to represent elemental, substructural, or monomer units. In [103][104], a linear-string GA is also used to search for combinations of molecular fragments encoded as linear strings. In [18], an approach is proposed to use GP with molecular design encoded first as trees. These encoding schemes impose different constraints on the EA operations so as to ensure the mimicking of the chemical reactions that take place when components and bonds between molecules are exchanged during, say, the crossover process. If these EA based techniques are used, post-processing is required to convert the linear-strings and trees to molecular designs in the form of molecular graphs.

For EvoGraph to carry out its tasks, a molecular design is encoded directly in the form of a *molecular graph*. A molecular graph is a connected, undirected graph representation of the structural formula of a chemical compound. It can be represented as  $G(V, E)$  where  $V$  is a set of vertex labels and  $E$  is a set of edge labels. A molecular graph

can therefore be represented as a labeled graph with vertices corresponding to the atoms of a compound and edges corresponding to the types of chemical bonds between atoms.

A hydrogen-depleted molecular graph is a molecular graph with hydrogen vertices deleted. The target molecules that EvoGraph evolves are organic molecules that are made up of carbon and hydrogen atoms and other components. Since carbon-hydrogen bond can be assumed to fill up the valence when there is no other bond connected to a carbon atom in an organic molecule, the use of hydrogen depleted graphs has been popular and EvoGraph also makes use of it.

EvoGraph operates on the adjacency matrices of the molecular graphs. An adjacency matrix represents graph nodes and their connectivity at the same time. Though these matrices can also be encoded in linear-string chromosomes which simple GAs can operate on by concatenating the rows of a graph adjacency matrix to form a linear array, it should be noted that this can be computationally clumsy. If linear-string chromosomes are used, the connectivity between nodes cannot be read directly. Furthermore, special decoding is required to convert the linear-string chromosome back into a molecular graph.

An additional advantage of the adjacency matrix encoding scheme is that an effective crossover operator can be relatively easily implemented with it. The “repairing” of a matrix after crossover to ensure connectivity can also be more easily implemented with adjacency matrices as described in Section 4. Comparing with other encoding schemes, molecular graphs encoded in adjacency matrices suit much better the special characteristics of molecular topologies and the reproduction operators EvoGraph adopts.

Given a molecular graph represented as,  $G(V, E)$ , where  $V$  is a set of vertex labels and  $E$  is a set of edge labels, we can construct its adjacency matrix in such a way that if

there exists a connection from vertex,  $v_i \in V$  to vertex  $v_j \in V$  in the graph, then the entry of the cell at the  $i^{th}$  row and  $j^{th}$  column,  $c_{ij}$  is set to 1, otherwise, if there is no connection between them, it is set to 0. For the case of a graph where each node and edge are labeled with specific attribute values, then  $c_{ij}$  can take on such values instead of 1 or 0. For example, for representing a double bond in a molecule, the value for  $c_{ij}$  can be set to '2'.

It should be noted that either an upper or lower triangular matrix or a symmetric matrix can be used to represent such graph. An upper triangular matrix will be used to illustrate the crossover and mutation operators. Symmetric matrices will be used for the purpose of deriving quantitative descriptors of molecular graphs.

### **8.3 The Fitness Function**

To determine the fitness value of a molecular graph, EvoGraph takes into consideration the various feature descriptors of a molecule encoded in its graph. These molecular feature descriptors include the weights of the atoms, their composition, the topology and adjacency of different atoms and bonds within the molecule. Since no single descriptor can completely describe all features of a molecule, EvoGraph uses different combinations of feature descriptors in different fitness functions so that target molecules of various complexities can be most comprehensively described. In the following, we describe these different molecular feature descriptors that EvoGraph uses for fitness evaluation.

### 8.3.1 Molecular Topology Descriptors

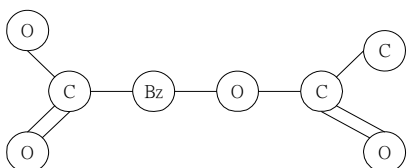
There have been more than 400 topological indices available [64]. Though none of the topological indices can capture all the topological characteristics of a graph, each topological index has its own merit. The Ivanciuc-Balaban operator (IB) [65] is computed with vertex invariants derived from symmetric molecular matrix. Because EvoGraph encodes hydrogen depleted molecular graphs into matrices to carry out evolution, IB is suitable for specifying characteristics of the target molecule for the evolution process. According to [65], IB is defined as

$$IB(M, w, G) = \frac{|E|}{\mu + 1} \sum_{E(G)} [VS_i(M, w, G) \times VS_j(M, w, G)]^{-1/2} \quad (1)$$

where  $G$  is the hydrogen depleted molecular graph,  $M$  is symmetric molecular weighted adjacency matrix,  $|E|$  is the number of edge of  $G$ ,  $w$  is the number of covalent bonds of an atom with other atoms in  $G$ ,  $VS_i(M, w, G)$  and  $VS_j(M, w, G)$  denote the vertex sums of the two adjacent nodes  $v_i$  and  $v_j$  that are incident with an edge  $e_{ij}$  in the molecular graph  $G$ , the summation goes over all edges from the edge set  $E(G)$ , and  $w$  is the weighting scheme,  $\mu$  is the cyclomatic number  $|E| - |V| + 1$  where  $|V|$  is the number of nodes in  $G$ . IB can be computed easily from a symmetric molecular matrix.

An example of IB for aspirin molecule is illustrated below. An aspirin hydrogen depleted molecular graph is shown in Figure 43(a). The symbol for benzene ring is 'Bz'. Its corresponding symmetric molecular matrix is shown in Figure 43(b). The nodes are indexed by atomic weights of atoms. Hence, carbon is labeled with 12, oxygen with 16,

and benzene ring with 72. The number of bonds between atoms is inserted as weights of edges in the matrix.



	12	12	12	16	16	16	16	72
12	0	0	0	2	1	0	0	1
12	0	0	1	0	0	1	2	0
12	0	1	0	0	0	0	0	0
16	2	0	0	0	0	0	0	0
16	1	0	0	0	0	0	0	0
16	0	1	0	0	0	0	0	1
16	0	2	0	0	0	0	0	0
72	1	0	0	0	0	1	0	0

(a) aspirin molecular graph

(b) symmetric molecular matrix of aspirin molecular graph

**Figure 53:** Aspirin molecular graph and symmetric molecular matrix

$$|E|_{aspirin} = 7$$

$$\sum_{E(G)} [VS_i(M, w, G) \times VS_j(M, w, G)]^{-1/2}$$

$$= (4 \times 2)^{\frac{1}{2}} + (4 \times 1)^{\frac{1}{2}} + (4 \times 2)^{\frac{1}{2}} + (4 \times 1)^{\frac{1}{2}} + (4 \times 2)^{\frac{1}{2}} + (4 \times 2)^{\frac{1}{2}} + (2 \times 2)^{\frac{1}{2}} = 2.9142$$

$$\mu = 7 - 8 + 1 = 0$$

$$IB_{aspirin}(M, w, G) = \frac{7}{1} \times 2.9142 = 20.3995$$

IB contains topological information of a molecular graph. But it does not contain information on the atoms. The target molecules to be evolved in this section are covalent molecules. In order to describe a molecule more accurately, composition of atomic weights (W), valences of atoms (V), total number of carbon atoms (NC), number of benzene rings (NBZ) and product of atomic weights of adjacent atoms from the first to



the second layer of adjacencies (AN, AN2) are used in the fitness function for the graph evolution in addition to IB.

Also, the eigenvector or graph spectrum (**S**), which works on symmetric adjacency matrix of a molecular graph (where all non-single bonds are counted as single bonds so as to describe the adjacencies only) is used to describe topology of a molecule. It is a graph invariant to specify the topology of a molecule [66]. To do so, let **A** be a symmetric adjacency matrix with eigenvalue  $\lambda$  and eigenvector **e**. By definition of eigenvector,  $\mathbf{Ae}=\lambda\mathbf{e}$ . Let **B** be a permutation of **A**,  $\mathbf{A} = \mathbf{PBP}^T$ , where **P** is a permutation matrix and  $\mathbf{PP}^T = \mathbf{I}$ , **I** is the identity matrix. Hence,  $\mathbf{Ae} = \mathbf{PBP}^T\mathbf{e} = \lambda\mathbf{e} \Rightarrow \mathbf{P}^T\mathbf{PBP}^T\mathbf{e} = \lambda\mathbf{P}^T\mathbf{e} \Rightarrow \mathbf{B}(\mathbf{P}^T\mathbf{e}) = \lambda(\mathbf{P}^T\mathbf{e})$ . Let  $\mathbf{S} = \mathbf{P}^T\mathbf{e} \Rightarrow \mathbf{BS} = \lambda\mathbf{e}'$ . **S** is the eigenvector of **B** and  $\lambda$  is also its eigenvalue. The set of elements in **e** is the same as **e'** because  $\mathbf{P}^T$  only affects the order of the elements but not their values. Therefore, the eigenvalues of a symmetric adjacency matrix is invariant to matrix permutation.

The closeness of topologies between two graphs is compared by the root mean square distance between their corresponding graph spectrums. Let vectors  $\mathbf{S}_A = [a_1, a_2, \dots, a_n]^T$  and  $\mathbf{S}_B = [b_1, b_2, \dots, b_n]^T$  be the graph spectrums  $G_A$  and graph  $G_B$  respectively. The elements in  $\mathbf{S}_A$  and  $\mathbf{S}_B$  are eigenvalues arranged in ascending order such that  $a_1 \leq a_2 \leq \dots \leq a_n$  and  $b_1 \leq b_2 \leq \dots \leq b_m$ .

If the sizes of *A* and *B* are equal, the distance between  $G_A$  and  $G_B$  is defined as

$$d(G_A, G_B) = \sqrt{\sum_{i=1}^n |a_i - b_i|^2} \quad (2)$$

If the sizes of  $A$  and  $B$  are unequal, the size of the shorter spectrum is increased by adding dimensions to bring it to match the other spectrum and zero elements are assigned to the new created spaces. The distance can then be computed with the same formula.

Like other topology indices,  $\mathbf{S}$  cannot completely describe the topology of a graph but it is an approximation of the topology expressed in a linear vector of real numbers. The graph topology corresponding to a graph spectrum may not be unique. Some non-isomorphic graphs have the same spectrum and are known as cospectral graphs.  $\mathbf{S}$  is used together with the IB operator and other atom adjacency descriptors to specify a target molecule for the evolutionary process.

### 8.3.2 Atom Adjacency Descriptors

Given a molecule's topology, adjacencies between atoms are required to specify the relative locations of atoms within the molecule. In order to specify the adjacency between any two atoms, a quantity is defined below as adjacent node index (AN) as follow.

$$AN = \sum_{i,j} \frac{1}{\sqrt{n_i n_j}} \quad (3)$$

where node  $i$  and node  $j$  are adjacent to each other,  $n_i$  is the node index of node  $i$ , and  $n_j$  is the node index of node  $j$ .

Sometimes two molecules with the same topologies and AN may have different molecular structures because the same adjacent pairs of atoms may exchange locations within the same molecule without varying the AN. A specification of second layer of atom adjacencies is devised as second layer adjacent node index (AN2) as follow.

$$AN2 = \sum_{i,j} \frac{1}{[P(A_i)P(A_j)]^{[N(A_i)+N(A_j)]}} \quad (4)$$

where node  $i$  and node  $j$  are adjacent to each other,  $A_i$  is the set of node indices of nodes adjacent to node  $i$ ,  $A_j$  is the set of node indices of nodes adjacent to node  $j$ ,  $P(A_i)$  is the product of node indices in  $A_i$ ,  $P(A_j)$  is the product of node indices in  $A_j$ ,  $N(A_i)$  is the number of elements in  $A_i$ , and  $N(A_j)$  is the number of elements in  $A_j$

In addition to using atom indices, the number of bonds attached to adjacent atoms can be used to specify the adjacencies of bonds within a molecule. The adjacent node-bond index (ANB) is defined as follow.

$$ANB = \sum_{i,j} \frac{1}{(v_i v_j e_{ij})^{\frac{1}{3}}} \quad (5)$$

where node  $i$  and node  $j$  are adjacent to each other,  $v_i$  is the number of bonds attached to node  $i$ ,  $v_j$  is the number of bonds attached to node  $j$ , and  $e_{ij}$  is the number bonds between node  $i$  and node  $j$ .

The topology and atom adjacency descriptors in this section are used in different combinations in the fitness functions to specify the target molecules. The combinations depend on features and complexity of the molecule to be specified.

### 8.3.3 Fitness Functions for Evolution of Molecules

EvoGraph makes use of different fitness functions to evolve molecules with different complexities. These fitness functions take into consideration a set of basic features that characterize the target molecule to be evolved. These features include the molecular weights of the hydrogen depleted molecules (W), the number of benzene rings (NBZ), the total weight of carbon (CW), the sum of degree of each atomic component exceeding its valence (V), and other descriptors including IB, S, AN, AN2, and ANB.

As none of these descriptors can completely describe all the features of a molecule, different fitness functions are used by EvoGraph with different combinations of these descriptors depending on the complexity of the target molecule concerned. As the fitness function is a measure of closeness to the target, it reflects how much deviation there is in the quantities of these descriptors from the target molecule. When a particular target descriptor is used in a fitness function  $f_k$ , we place a ‘Y’ in the corresponding entry in the table, Table 17 below.

fitness function	W	NBZ	NC	V	IB	S	AN	AN2	ANB
$f_1$	Y	Y	Y	Y	Y	-	Y	-	-
$f_2$	Y	Y	Y	Y	Y	Y	-	Y	-
$f_3$	Y	Y	Y	-	-	Y	-	-	-
$f_4$	-	-	-	-	-	-	Y	Y	-
$f_5$	-	-	-	Y	-	-	-	-	Y

**Table 17:** Summary of fitness functions and their target properties

$$f_k = \frac{1}{\sum_i 1.1^{|\delta_i(t)|}} \quad (6)$$

where  $k \in \{1,2,3,4,5\}$ ,  $t$  is the target value of descriptor  $i$ , and  $|\delta_i(t)|$  is the absolute deviation of the value of a descriptor  $i$  to its target value. According to Table 17,

- for  $f_1$ ,  $i \in \{W, NBZ, NC, V, IB, AN\}$
- for  $f_2$ ,  $i \in \{W, NBZ, NC, V, IB, S, AN2\}$
- for  $f_3$ ,  $i \in \{W, NBZ, NC, S\}$
- for  $f_4$ ,  $i \in \{AN, AN2\}$
- for  $f_5$ ,  $i \in \{V, ANB\}$

In order to ensure that all  $|\delta_i(t)|$ 's are linear, the absolute deviation from the target spectrum,  $|\delta_s(t)|$ , is the root mean square Euclidean distances  $d(G_A, G_B)$  given in (2). When a criteria is satisfied, it's corresponding  $|\delta_i(t)|$  equals zero. When  $|\delta_i(t)| = 0$  for all  $i$ ,  $f_k = 1$ . The target is reached and the evolution converges to unity.

The more descriptors that are used, the more computational resources are required when computing fitness values. The most computational intensive fitness function is  $f_2$ . It should be noted therefore that computational resource may have to be compromised when descriptors are added to evolve more complicated molecules.

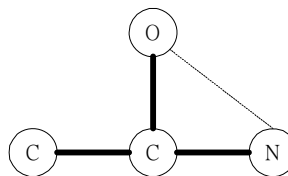
Small molecular graphs with fewer nodes and less complicated molecular structures are easier to describe and they can be evolved by using one fitness function. For larger molecular graphs and more complicated molecular structure, more than one stage evolution is required. Different fitness functions are used at different stages of evolution to approach the target molecule. They are shown in Section 8.4.

### 8.3.4 An Illustrative Example in Drug Design

To illustrate how EvoGraph works with chemical compounds, we represent molecular structure as hydrogen depleted graphs with nodes of graph representing the atom and edges representing the bonds. Let the symbol for carbon be 'C', oxygen be 'O', nitrogen be 'N', and benzene ring be 'Bz'. In the molecular matrix, the nodes are indexed by the atomic weights of the atoms for the convenience of calculation of fitness values. Hence, carbon is labeled with 12, oxygen with 16, nitrogen with 14, and benzene ring with 72. The number of bonds between atoms is inserted as weights of edges in the matrix.

The initialization process generates a population of *molecular graphs* composed of elements bonded together. To generate a molecular graph, a few elements are selected at random. A spanning tree is then added to connect all elements together followed by addition of other bonds at random. An example of molecular graph initialized is shown below.

	12	12	16	14
12	-	1	0	0
12		-	1	1
16			-	1
14				-



(a) adjacency matrix of *molecular graph* with spanning tree bonds indicated as '1's and randomly added bond as '1'

(b) *molecular graph* with spanning tree indicated as bold line and randomly added bond as dotted line

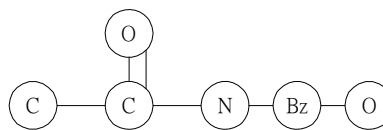
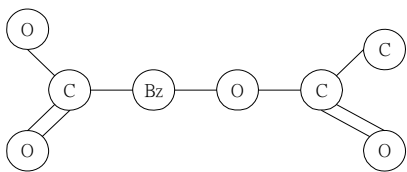
**Figure 54:** Example of *molecular graph* generated by initialization

The example on *random crossover* of molecular graph is indicated below. Consider two common molecules aspirin and tylenol.

1. Construct hydrogen depleted *molecular graphs* and adjacency matrices.
2. Permute the order of nodes in the adjacency matrices at random. The results are shown in Figure 55.

	12	12	12	16	16	16	16	72
12	-	0	0	2	1	0	0	1
12		-	1	0	0	1	2	0
12			-	0	0	0	0	0
16				-	0	0	0	0
16					-	0	0	0
16						-	0	1
16							-	0
72								-

	12	12	16	16	14	72
12	-	1	0	0	0	0
12		-	0	2	1	0
16			-	0	0	1
16				-	0	0
14					-	1
72						-



(a) aspirin

(b) tylenol

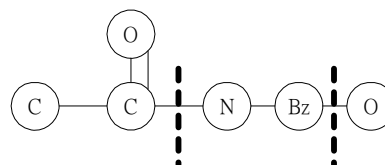
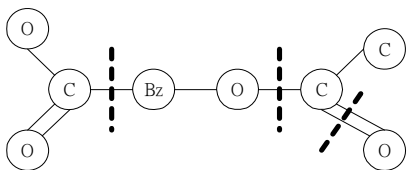
**Figure 55:** *Molecular graph* and adjacency matrices of aspirin and tylenol

3. A cut line is inserted at random across each of the molecular matrix as shown in Figure

56. The corresponding cut lines in the molecular graphs are shown.

	12	12	12	16	16	16	16	72
12	-	0	0	2	1	0	0	1
12		-	1	0	0	1	2	0
12			-	0	0	0	0	0
16				-	0	0	0	0
16					-	0	0	0
16						-	0	1
16							-	0
72								-

	12	12	16	16	14	72
12	-	1	0	0	0	0
12		-	0	2	1	0
16			-	0	0	1
16				-	0	0
14					-	1
72						-



(a) aspirin

(b) tylenol

**Figure 56:** Random cut line imposed on molecular matrices of aspirin and tylenol

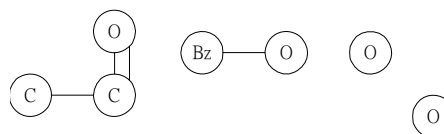
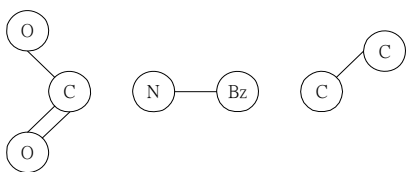


4. Exchange cut out submatrices.

5. Scan and delete invalid edges

	12	12	12	16	16	14	72
12	-	0	0	2	1	0	0
12		-	1	0	0	0	0
12			-	0	0	0	0
16				-	0	0	0
16					-	0	0
14						-	1
72							-

	12	12	16	16	16	16	72
12	-	1	0	0	0	0	0
12		-	0	2	0	0	0
16			-	0	0	0	0
16				-	0	0	0
16					-	0	1
16						-	0
72							-



(a)

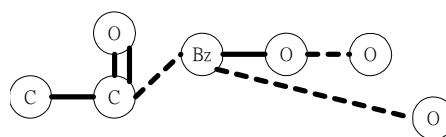
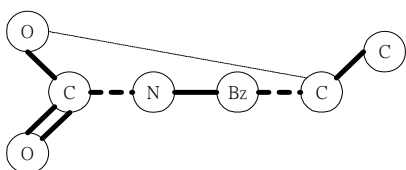
(b)

**Figure 57:** Swapping of subgraphs created by the cut and deletion of invalid edges

5. Superimpose a spanning tree at random over the existing edges on each graph after swapping and add edges at random to create the offspring.

	12	12	12	16	16	14	72
12	-	0	0	2	1	1	0
12		-	1	0	1	0	1
12			-	0	0	0	0
16				-	0	0	0
16					-	0	0
14						-	1
72							-

	12	12	16	16	16	16	72
12	-	1	0	0	0	0	0
12		-	0	2	0	0	1
16			-	0	1	0	1
16				-	0	0	0
16					-	0	0
16						-	1
72							-



(a)

(b)

**Figure 58:** Embed spanning tree at random to each offspring graph after swapping. Superimposed spanning tree in '1's in the adjacency matrices. The embedded spanning trees are shown in bold lines in the graphs. The new edges connecting the swapped subgraphs are shown in dotted line.

Mutations are shown by using the example of aspirin molecule below. *Number-of-Node* mutation involves addition or deletion of one element in a molecule. *Number-of-Edge* mutation involves addition or deletion of one bond in a molecule, and *Swap-Vertex* mutation involves swapping of two elements in a molecule. They are shown in the molecular matrices in Figures 59, 60 and 61 respectively. The changes are highlighted by italics and bold letters.

	12	12	12	16	16	16	16	72
12	-	0	0	2	1	0	0	1
12		-	1	0	0	1	2	0
12			-	0	0	0	0	0
16				-	0	0	0	0
16					-	0	0	0
16						-	0	1
16							-	0
72								-

	12	12	12	16	16	16	72
12	-	0	0	2	1	0	1
12		-	1	0	0	1	0
12			-	0	0	0	0
16				-	0	0	0
16					-	0	0
16						-	1
72							-

(a) select at random an element O to be deleted in aspirin (b) delete O from aspirin

	12	12	12	<u>12</u>	16	16	16	16	72
12	-	0	0	<u>0</u>	2	1	0	0	1
12		-	0	<u>0</u>	0	0	0	0	1
12			-	<u>1</u>	0	0	1	2	0
<u>12</u>				-	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>
16					-	0	0	0	0
16						-	0	0	0
16							-	0	1
16								-	0
72									-

(c) add at random one carbon to aspirin

**Figure 59:** The *Number-of-Node* mutation operator illustrate

	12	12	12	16	16	16	16	72
12	-	0	0	2	1	0	0	1
12		-	1	0	0	1	2	0
12			-	0	0	0	0	0
16				-	0	0	0	0
16					-	0	0	0
16						-	0	1
16							-	0
72								-

	12	12	12	16	16	16	16	72
12	-	0	0	2	1	0	0	1
12		-	1	0	0	1	<u>1</u>	0
12			-	0	0	0	0	0
16				-	0	0	0	0
16					-	0	0	0
16						-	0	1
16							-	0
72								-

(a) select at random a bond C-O to be deleted in aspirin

(b) delete one C-O bond from aspirin

	12	12	12	16	16	16	16	72
12	-	0	0	2	1	0	0	1
12		-	1	0	0	1	2	0
12			-	0	0	0	0	<b>0</b>
16				-	0	0	0	0
16					-	0	0	0
16						-	0	1
16							-	0
72								-

	12	12	12	16	16	16	16	72
12	-	0	0	2	1	0	0	1
12		-	1	0	0	1	2	0
12			-	0	0	0	0	<u>1</u>
16				-	0	0	0	0
16					-	0	0	0
16						-	0	1
16							-	0
72								-

(c) select at random one bond C-Bz to be added to aspirin

(d) add one C-Bz bond to aspirin

**Figure 60:** The *Number-of-Edge* mutation operator illustrated

	12	12	12	16	16	16	16	72
12		0	0	2	1	0	0	1
12			1	0	0	1	2	0
12				0	0	0	0	0
16					0	0	0	0
16						0	0	0
16							0	1
16								0
72								

	12	12	16	16	16	12	16	72
12	-	0	0	2	1	0	0	1
12		-	1	0	0	1	2	0
16			-	0	0	0	0	0
16				-	0	0	0	0
16					-	0	0	0
12						-	0	1
16							-	0
72								-

(a) select a pair of elements C and O in aspirin to be swapped (b) swapping C and O

**Figure 61:** The *Swap-Node* mutation operator illustrated

To illustrate how the fitness value of a molecular graph can be evaluated, the *molecular graph* in Figure 58(a) is used for evaluation of fitness. The target molecule for reference in the illustration is assumed to be aspirin in Figure 53. The fitness function used is  $f_l$ . The descriptors used by  $f_l$  are W, NBZ, CW, V, IB, AN.

	W	NBZ	CW	V	IB	AN
Target value of descriptors of aspirin ( $t$ )	172	1	108	0	20.3995	0.4355
Value of descriptors of <i>molecular graph</i> in Figure 58(a) ( $u$ )	154	1	108	0	10.3408	0.4425
$ \delta_i(t)  =  t - u $	18	0	0	0	10.0587	0.007

**Table 18:** Illustration of fitness value calculation using  $f_l$

From (6), the fitness is therefore  $f_1 = \frac{1}{1.1^{(18+0+0+0+10.0587+0.007)}} = 0.069 < 1$

## 8.4 Experiments on Evolution of Molecules

To evaluate the performance of EvoGraph, it is used to see if it can discover known drug molecules. The hydrogen depleted molecular graphs and the symmetric adjacency matrices of these molecules are shown in Appendix 5.

### 8.4.1 Initialization and Evolution Parameters

An initial population of 50 symmetric molecular matrices containing node labels of the target molecular graph is generated at random. Steady State Reproduction and roulette wheel selection is used as described in Section 4.5.

There has not been a clear theory to guide the parameterization and design of evolutionary algorithms [71]. In GA, crossover and mutation rates are often chosen by trial and error to serve their purpose. In EP, crossover is not used at all. Crossover is the major evolution operator for EvoGraph while mutation is the minor operator to prevent the population from converging to local optimum of fitness less than unity. But the relative proportion of mutation to crossover is higher in comparing with conventional GA to increase the role of random search. The numbers of crossovers and ‘node mutation’ are fixed at 10 and 1 respectively. The numbers of ‘bond mutation’ and ‘swap node mutation’ are adjusted between 1 and 5 to in each trial to minimize the number of generations to convergence.

### 8.4.2 Experimental Setup and Results

Graph topologies can be partitioned into tree topologies and non-tree topologies. The latter consist of rings. In comparing with tree topologies, the cyclical nature of ring structure in a molecule causes more overlapping of first layer atom adjacencies and bond

adjacencies. More descriptors are required to accurately describe the target molecules with rings. The target molecules we intend to design with EvoGraph can be divided into 2 groups. Members of the first group have tree topologies. They include tylenol, adrenaline, aspirin, and ibuprofen. Members of the second group have ring structures. They include nicotine, caffeine, and dilantin. Descriptors for tree-structure molecules, and hence their fitness functions, are relatively simple when comparing with those of graphs with ring structures. The first group of target molecules is evolved using the same fitness function.

The attempt to include all descriptors in Table 17 in a single fitness function to evolve the second group of complex molecules causes prolongation of convergence time and requires heavy computational resources. The evolutionary processes could not converge within 3000 generations in our experiments when we attempted to do so. The strategy we adopted in our experiments was therefore to break down the search in stages with each stage deploying different reproduction operators and feature descriptors for fitness evaluation so that some properties of the molecule being designed can be attained at one stage and the resulting molecule can be passed over to the next stage to evolve into another molecular with another set of properties.

For EvoGraph, the molecule obtained in one stage does not destroy the results of the previous stages. For more complex target molecules, they are evolved in stages with EvoGraph using different fitness functions at different stages.

The first stage finds the right atomic components and topology of a molecule using  $f_3$  as the fitness function and RGC as operators. This first stage targets to search for the right atomic components and topology of the target molecule. With the right atomic components and topology found, the second and third stages use mutation to swap the

atomic components and mutate bonds within a molecule to their right places. The second stage deploys  $f_4$  as the fitness function and the *Swap-Node* mutation operator. It targets at the searching of the right adjacencies between atomic components. For the third stage, the fitness function of,  $f_5$  and the *Number-of-Edge* mutation operator is used. This aims at the search of the right bond adjacencies. The topology and the atomic components evolved in the first stage are therefore not affected by the mutations in the second and third stage.

For the first group of drug molecules we used in our experiment, EvoGraph was applied to search the 4 target molecules in Appendix 5 using the fitness function  $f_l$ . Ten experiments are conducted for each target molecule with the evolution parameters as described in this section. The best results for each target molecule are selected. EvoGraph converged in all experiments in less than 2000 generations. However, the simplest 3 of the 4 molecules, tylenol, adrenaline, and aspirin converge and produce the exact target molecules whilst an isomer is evolved for ibuprofen. The result of the first stage experiments is shown in Table 19.

Molecule	Number of random crossover per generation	Number of Number-of-Node mutation per generation	Number of Number-of-Edge mutation per generation	Number of Swap-Node mutation per generation	Number of generations of convergence to unity	Exact match to target molecule or isomer
tylenol	10	1	5	2	270	exact match
adrenaline	10	1	2	2	519	exact match
aspirin	10	1	5	2	1937	exact match
ibuprofen	10	1	5	2	1351	isomer

**Table 19:** First group experimental result using  $f_l$  as fitness function



The fitness function  $f_1$  controls only the most basic features of a molecule. W, NBZ, CW, and V control the composition of atoms, atomic weights and valence of atoms within the molecule; IB and AN, control the first layer adjacencies between bonds and atoms in a molecule. Simple molecules are evolved based on these descriptors. For more complex molecules, control on second layer adjacencies and more stringent control of the topology is required.

To improve the result, a more computational intensive fitness function,  $f_2$  is used in place of  $f_1$  to evolve ibuprofen. In  $f_2$ , the more computational intensive second layer adjacency descriptor, AN2 replaces the single layer adjacency descriptor, AN, in  $f_1$ . The topology descriptor, graph spectrum S, is also added to enhance the overall topology description. IB is added to enhance the bond description. With the use of  $f_2$  and the same evolution operator proportions, the exact match is achieved at the 2289<sup>th</sup> generation. The result is summarized in Table 20.

Molecule	Number of random crossover per generation	Number of Number-of-Node mutation per generation	Number of Number-of-Edge mutation per generation	Number of Swap-Node mutation per generation	Number of generations of convergence to unity	Exact match to target molecule or isomer
ibuprofen	10	1	5	2	2289	exact match

**Table 20:** Evolving ibuprofen using  $f_2$  as fitness function

Instead of satisfying all descriptors at one go, which would require rather long convergence time, the second group of target molecules was allowed to evolve in 3 stages. The components and topology of a molecule is first evolved in stage 1 (using  $f_3$  with descriptors W, NBZ, CW, and S). This is then followed by adjacencies of atoms in stage

2 (using  $f_4$  with descriptors AN and AN2), and finally the valence of atoms and bond types in stage 3 (using  $f_5$  with descriptors V and ANB).

In the first stage of the second group experiments, the evolutionary operators used is the same as first group experiments except *Swap-Node* mutation because atom adjacencies are not dealt with at this stage. Ten experiments are conducted for each target molecule with 10 crossovers and 1 node mutation in each generation. The number of *Number-of-Edge* mutations in each generation is selected between 1 and 5 to minimize the number of generations on convergence. The fitness function used is  $f_3$ . The best results for each target molecule are selected. The correct topologies are evolved when fitness converge to unity. There is no cospectral graph being evolved. The result is summarized in Table 21.

Molecule	Number of random crossover per generation	Number of <i>Number-of-Node</i> mutation per generation	Number of <i>Number-of-Edge</i> mutation per generation	Number of <i>Swap-Node</i> mutation per generation	Number of generations of convergence to unity	Exact match to target topology or cospectral graph
nicotine	10	1	5	NA	1566	exact match
caffeine	10	1	2	NA	1203	exact match
dilantin	10	1	5	NA	1183	exact match

**Table 21:** Second group stage 1 experiments using fitness function  $f_3$

In stage 2, only *Swap-Node* mutation is performed on the molecules evolved in stage 1 to achieve the right atom adjacencies. Fitness function  $f_4$  is used with the number of *Swap-Node* mutation per generation fixed at 10. The result is summarized in Table 22.

Molecule	Number of <i>Swap-Node</i> mutation per generation	Number of generations of convergence to unity
nicotine	10	63
caffeine	10	118
dilantin	10	679

**Table 22:** Second group stage 2 experiments using fitness function  $f_4$

In stage 3, the *Number-of-Edge* mutation operator is adjusted to mutate bond numbers between connected atomic components in a molecule without changing its topology. It is performed on the molecules evolved in stage 2 with the number of *Number-of-Edge* mutation operator per generation fixed at 10. The fitness function,  $f_5$  that targets on bond description is used with the number of *Number-of-Edge* mutation operator per generation fixed at 10. The result is summarized in Table 23.

Molecule	Number of <i>Number-of-Edge</i> mutation per generation	Number of generations of convergence to unity
nicotine	10	58
caffeine	10	525
dilantin	10	438

**Table 23:** Second group stage 3 experiments using fitness function  $f_5$

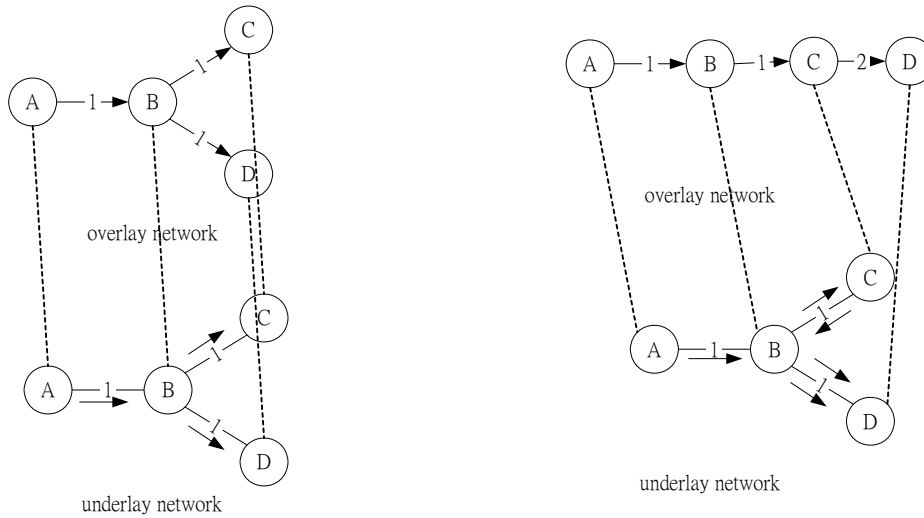
In the experiments on stage 2 and 3 experiments, the number of generations to convergence increases with the degree of symmetry of the molecules. The least symmetrical is nicotine and the most symmetrical is dilantin. Symmetry of a molecule renders the descriptors of atom adjacencies and bond adjacencies inefficient. Though the problem may be overcome by incorporating symmetry descriptors in the fitness function, it may further increase computation resource required for the fitness function.

---

## Peer-to-Peer Overlay Network Design

---

Overlay network is a virtual network formed by end hosts atop an underlay physical network to implement communication services for distributed applications. The problem of flooding of queries and responses amongst the end hosts in the overlay network is raised in [72][73]. There is a substantial wastage of resource on flooding of overlay network as illustrated in the example in Figure 62. Two overlay network topologies with unit cost of links, over the same underlay network are illustrated in Figure 62. Node 'A' is the source node in the network. In Figure 62(a), the overlay network cost is 3 while in Figure 62(b) the cost is 4. Link BC and BD in Figure 62(b) are being traversed two times instead of once as in Figure 62(a). This is due to the topology mismatch between the overlay network and the underlay network in Figure 62(b). Similarly, the same problem occurs between the routing paths over the logical layer P2P overlay network as revealed in [75]. Hence, the choice of routing topology determines the efficiency of distribution of information. P2P networks that rely on broadcast and back propagation from the source node to the neighboring nodes, such as Gnutella [74], will be hampered tremendously by flooding caused by inappropriate overlay network topologies.



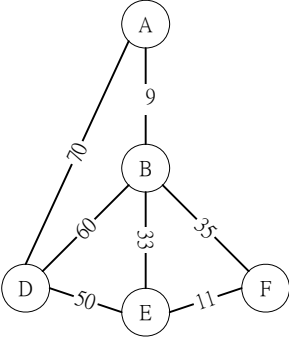
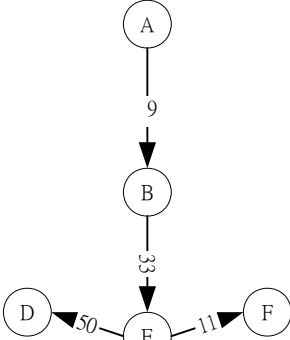
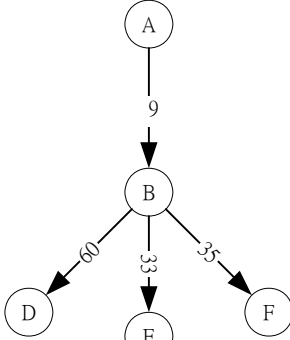
(a) topology match

(b) topology mismatch

**Figure 62:** Comparison of 2 different overlay networks on the same underlay network.

Minimum Spanning Tree (MST) based algorithms has been used to minimize topology mismatches [4][22]. However, a MST may not minimize the total cost of flow from the source node to the destinations in the system. Without considering topology mismatch, the solution to minimize the cost of flow from the source to the destinations is to construct the Shortest Path Tree (SPT). Knowing this, some MST based algorithms modify the MST topology to minimize partially the cost of flow at the expense of topology match [21]. The solutions are somewhere in between the MST and SPT. There have not been any clear criteria on the degree of compromise between MST and SPT. Our intention is to propose the criteria and corresponding algorithm to search for solutions to meet them. Comparison between MST and SPT is illustrated by an example in Figure 63. Two different overlay network topologies over the same underlay network are compared against each other. Figure 63(a) shows the underlay network with cost of flow labeled on the edges. Figure 63(b) is the MST and Figure 63(c) is the SPT over the

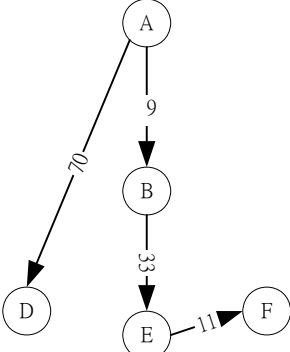
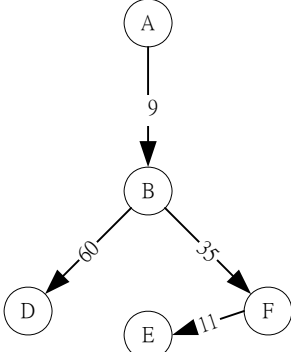
underlay network. We define the Total Edge Cost (TEC) as the sum of cost of all edges in a spanning tree, the Total Path Cost (TPC) as the sum of all path costs from the source to the destinations, and Total Cost (TC) as the sum of TEC and TPC. The cost of flow between any two nodes, say A and B, is denoted by  $d(AB)$ . The TEC of MST is denoted by  $TEC_{MST}$ . Other denotations are derived similarly. They are  $TEC_{SPT}$ ,  $TPC_{MST}$ ,  $TPC_{SPT}$ ,  $TC_{MST}$ , and  $TC_{SPT}$ .

		
	$TEC_{MST}=9+33+50+11=103$ $d(AB)=9$ $d(AD)=92$ $d(AE)=42$ $d(AF)=53$ $TPC_{MST}=196$ $TC_{MST}=TEC_{MST}+TPC_{MST}=299$	$TEC_{SPT}=9+60+33+35=137$ $d(AB)=9$ $d(AD)=69$ $d(AE)=42$ $d(AF)=44$ $TPC_{SPT}=164$ $TC_{SPT}=TEC_{SPT}+TPC_{SPT}=301$
(a) underlay network	(b)MST overlay network	(c)SPT overlay network

**Figure 63:** Comparison of costs of flow of MST and SPT over the same underlay network

All algorithms on constructing MST aim at minimizing  $TEC_{MST}$ . There is a compromise on the minimization of TPC. For example,  $TPC_{MST}=196$  in Figure 63(b)  $>$   $TPC_{SPT}=164$  in Figure 64(c). On the other hand, construction of SPT aims at minimizing  $TPC_{SPT}$  and there is compromise on the minimization TEC.  $TEC_{SPT}=137$  in Figure 63(c) $>$  $TEC_{MST}=103$  in Figure 63(b). We assume the minimization of both TEC and TPC

are equally important. Our intention is to find a spanning tree atop a given underlay network with  $TEC+TPC=TC \leq \min(TC_{MST}, TC_{SPT})$ . We name this spanning tree Approximate Minimum Total Cost Tree (AMTCT). In this example,  $TC_{AMTCT} \leq \min(299, 301) = 299$  should be satisfied. There may be more than one AMTCT for a given underlay network. For example, two AMTCTs atop the underlay network in Figure 63(a) are illustrated in Figure 64. The edges in AMTCTs do not necessarily come from the MST or TEC. For example, in Figure 64(a) edge AD comes from the underlay network and it does not form part of the MST or TEC in Figure 64(b) and (c).

	
<p><math>TEC_{AMTCT1} = 123</math></p> <p><math>d(AB) = 9</math>  <math>d(AD) = 70</math>  <math>d(AE) = 42</math>  <math>\underline{d(AF) = 53}</math></p> <p><math>TPC_{AMTCT1} = 174</math></p> <p><math>TC_{AMTCT1} = TEC_{AMTCT1} + TPC_{AMTCT1} = 297 &lt; \min(299, 301)</math></p>	<p><math>TEC_{AMTCT2} = 115</math></p> <p><math>d(AB) = 9</math>  <math>d(AD) = 69</math>  <math>d(AE) = 55</math>  <math>\underline{d(AF) = 44}</math></p> <p><math>TPC_{AMTCT2} = 177</math></p> <p><math>TC_{AMTCT2} = TEC_{AMTCT2} + TPC_{AMTCT2} = 292 &lt; \min(299, 301)</math></p>
(a)	(b)

**Figure 64:** Two AMTCTs atop the underlay network in Figure 63(a) with corresponding costs of flow

The cost advantages of AMTCT1 and AMTCT2 over MST and SPT are summarized in Figure 65. In Figure 65(a), AMTCT1 is used to compare with the SPT and MST. When comparing with SPT, the AMTCT1 has a TEC saving of  $137 - 123 = 14$  and compensates

the loss of TPC at  $164-174=-10$  resulting in a net saving of 4. When comparing with MST, there is a cost saving of  $196-174=22$  for TPC which compensates the loss of cost value  $103-123=-20$  for TEC resulting in a net cost saving of 2. In Figure 65(b), similar comparison is made between the AMTCT2 and the SPT and MST. There is a net cost saving of in TC of 9 and 7 respectively. Hence, both AMTCT1 and AMTCT2 are acceptable solutions because there is an overall cost reduction from the original  $\min(\text{TC}_{\text{MST}}, \text{TC}_{\text{SPT}})=299$  in Figure 63(b) and (c).

	TPC	TEC	TC
SPT	164	137	301
AMTCT1	174	123	297
net	-10	14	4
% saving			1.3%

	TPC	TEC	TC
SPT	164	137	301
AMTCT2	177	115	292
net	-13	22	9
% saving			3%

	TPC	TEC	TC
MST	196	103	299
AMTCT1	174	123	297
net	22	-20	2
% saving			0.7%

	TPC	TEC	TC
MST	196	103	299
AMTCT2	177	115	292
net	19	-12	7
% saving			2.3%

**Figure 65:** Cost analysis of AMTCT1 and AMTCT2

The search for AMTCTs is a graph topology optimization problem which is NP hard. Graph based heuristic algorithms have been derived from conventional MST and SPT search algorithms [85] to tackle NP hard search on networks such as [81][122]. Their approach are based on probing of connections within a few hops from the source in the network base on the cost of flows of the paths traversed and cost of flow to the nodes available for probing in the next iteration. There is no optimization on the costs of flow and topology connecting the source and destinations. Some of them requires complete cost graph for all the nodes in the system to provide cost of flow between any two nodes in the system for probing. Some algorithms places emphasis on solving overlay network



queries and response flooding by minimization of topology mismatch between the overlay and underlay network but they face the problem of the heavy response traffic generated from the destinations back to the source which demand a shortest path between source and the destination. Most of these solutions [121] first find the MST that solves the former problem and then replace some high cost paths to the source by the shortest paths in the SPT to solve the latter. This results in a tree topology that lies somewhere between MST and SPT. Although these algorithms demonstrate the ability to reduce traffic congestion, the metric for optimization is not clear.

There are deterministic algorithms to search for a tree that lies between MST and SPT such that the TEC and TPC lies between certain cost bound making reference to the  $TEC_{MST}$  and  $TPC_{SPT}$ [123][124]. Their objective is not minimization of cost but to contain the cost within certain limit. Furthermore, the edges of the resulting tree should come from either the MST or TEC. Hence they cannot explore the use of new potential edges from the underlay network such as edge AD in Figure 64(a).

Graph based Evolutionary Algorithm (EA) is another approach to NP hard search on networks. At present, most of graph based EAs are designed for other problem domains such as GP for evolution of trees for computer programming [2] and EP for evolution of neural networks [4]. Other graph based EAs are designed with their encoding schemes to solve problems in their specific domains. They are not suitable for our search as reviewed in the next section.

In view of the lack of clear metric for optimization, our algorithm use  $TC=SPT+TPC$  as an objective of minimization which place equal weights on both minimization of cost of the spanning tree and costs of flow between the source and all the destinations.

EvoGraph is used to search for the AMTCTs atop an underlay network to minimize TC. EvoGraph treats networks as graphs and encodes them in cost matrices. Special evolution operators are designed to operate on the cost matrices to search for the optimal. It uses the underlay network with costs of flow between connected nodes as the backdrop to derive the initial population of graphs as well as subsequent crossover and mutation operation. Unlike other heuristic algorithms, there is no need to construct another layer of complete cost graph for the algorithm to operate. It can also explore the use of links in the underlay network that do not belong to the MST and SPT. The AMTCTs generated by EvoGraph do not need to attain the absolute cost minimum provided there is total cost savings to the MST and SPT. This eliminates the common problem of uncertainty of reaching the absolute minimum in heuristic search algorithms. This system can work with different groups of nodes in parallel throughout the Internet each having some knowledge on cost of flow with other nodes in its group via the packet exchange under Internet protocols.

The stability of the Internet and the efficiency of protocol transfer of routing information are not within the scope of this study. It is assumed the underlay network of the physical internet is relatively stable [125] for the source to generate to AMTCT and for the protocols to transmit routing information to the destination nodes. [126] measures the medium session duration for Gnutella and Napstar to be 60 minutes. With the understanding of time constraint on the maintenance of connections of nodes in a system, the average number of converging generations adopted for our evolutionary search experiments is kept below 1000. Underlay networks are generated at random starting from 8 nodes to 24 nodes at 2 nodes interval for EvoGraph to build the AMTCT atop.

The node degrees of the underlay network are generated at random between 2 and 8. This is adopted in the experiments to tally with the statistical samples in [127] where average out degrees for nodes in an Internet is 3. In our experiments, we observe the order of increase in number of generations to reach convergence as the size of network increases. This relation indicates the efficiency of EvoGraph on the time of convergence. The cost savings by AMTCT in comparing with the conventional MST and SPT on each of the underlay network are also observed. There can be more than one AMTCT generated for each underlay network. The number of AMTCTs generated in addition to the fittest AMTCT is observed with respect to the topological properties of the underlay network.

### **9.1 Previous Studies and the Problem**

Decentralized sharing of files at different locations in the P2P network such as Napster, Bit Torrent, Gnutella, Chord etc. [79] has been popular in recent years. This aroused a lot of studies on the solving the topology mismatch problem at the logical layer overlay P2P network, which directly affects the search efficiency as well as the scalability of the network [78]. Studies on both unstructured [75][76][77] and structured [80] P2P networks are conducted. The construction of minimum spanning tree (MST) from the source node to its logical neighbors is an efficient method on solving the topology mismatch problem. This method is included in the algorithms proposed in [73][75][81]. In addition to the aim of minimization of unnecessary traffic due to topology mismatch, minimization of total cost of traffic from source node to the logical neighbors is also important. By default, the Internet usually uses the shortest path between nodes. Studies on improving quality of service of overlay network are carried

out to optimize the cost of traffic from the source node to the neighboring nodes with optimal bandwidth distribution [82]-[85]. In these studies, finding the shortest paths from the source node to the neighboring nodes is a major component in their designs. The construction of shortest path tree (SPT) is included in their algorithms.

In the process of construction of MST for minimization of topology mismatch, the shortest path from source node to the destination nodes at the overlay network may be compromised. But minimization of path costs between the source and destinations are especially important for heavy response destinations because they generate a lot of traffic in response to queries from the source. To address this problem [121] adds a step to its algorithm of generating MST in 2 hops diameter from the source to replace the path from the source to a heavy response destination in the MST with the shortest path. Similarly, in [2],[75],[128]-[129] the algorithm first constructs MST within 2 hops from the source and then followed by replacing connections to nodes far away from the source by connections to nodes nearer to the source. In these cases, SPT served as a compensation of inadequacies of the MST. There is no clear objectives on the trade-offs between the two in the algorithms. Furthermore, these algorithms [2],[75],[121],[128]-[129] involve probing of connections from the source to neighboring nodes to construct the MSTs. A complete graph on cost of flow between the source and all the neighboring nodes has to be constructed to supply cost of the potential paths to be probed.

The SPT and MST can be found by conventional deterministic greedy algorithms on graph such as Dijkstra's algorithm and PRIM algorithm respectively [85]. There are graph based algorithms using deterministic approach on searching for tree in a graph with costs of flow that lie between  $TEC_{MST}$  and  $TPC_{SPT}$  by traversing the MST with edges

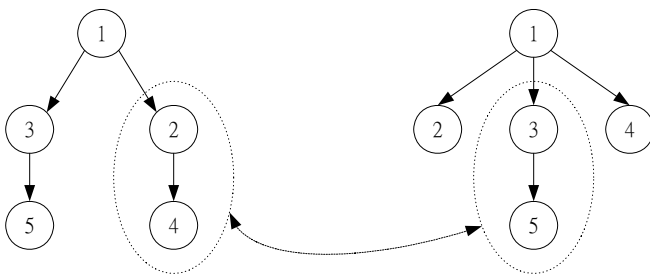
addition and deletion based on some criteria on maximum cost of flow from the source to the current node being traversed [123]-[124]. The resulting tree has the maximum cost of flow between the source and any destination bounded by a cost which is a multiple of the minimum path cost in the SPT and its TEC bounded by a cost which is a multiple of the  $TEC_{MST}$ . These algorithms construct trees with certain cost bounds based on SPT and MST. They do not search for the tree topologies that minimize TC and the trade off between SPT and MST is not clear. Furthermore, all the edges of the resultant tree comes from the MST or SPT. The potential edges outside the MST and SPT cannot be explored. Similarly, in [122] greedy algorithm is applied on SPT search with relaxation on letting the paths to pass through the destinations to find a tree somewhere between SPT and MST with low TEC and reasonable source-destination costs. The objective of cost minimization is not clear.

Our proposal on the search of AMTCTs is a network optimization problem. Evolutionary approach can be applied in this respect. There are many studies on using evolutionary approach to optimize network topologies such as encoding the adjacency matrices of graphs in linear chromosomes with application of genetic algorithm (GA)[91]. This approach has a high probability on producing disconnected graphs because connectivity of the parent graphs is not encoded in the chromosome. And the retention of useful links is not considered in the crossover process. Hence, its efficiency is hindered by the subsequent repair process. Post processing of the linear children chromosome after crossover and mutation is also required to convert it back into a graph.

Graph based EAs have been developed to search for special graph topologies required for different problems. EP[4] is designed to evolve neural network topologies and

corresponding weights of the edges using mutation alone. This algorithm is designed for incremental changes in a network by addition and deletion of an edge with subsequent repair if disconnected graph is produced. It does not have the benefit of capturing useful information from two parent networks in crossover process. The *Number-of-Edge* mutation in EvoGraph has the same mechanism.

GP [2] provides an evolutionary algorithm for tree topologies in form of subtree exchange between parent trees. This subtree exchange algorithm cannot maintain consistent nodes from the parent trees to the children tree which is the requirement for our search. This is because crossover point and hence the nodes to be swapped between parent trees are dictated by the tree topologies. This is illustrated by an example below. Node '1' is the source and nodes '2' to '5' are the destinations. The purpose is to produce children trees spanning all the nodes in the system. Each node should not appear twice in the children tree. The swapping of subtrees in Figure 66 will result in children trees having repetitive nodes. They are invalid solutions for both children.



**Figure 66:** Swapping of subtrees in GP producing invalid solutions for optimal routes

Prüfer numbers encoding is also used in other studies on evolution of trees. This is an elegant way of encoding trees in linear chromosome because its decoding is found in a constructive proof of Cayley's formula. Though Prüfer numbers encoding have been used

to represent trees in GA [90] and many studies use it as a heuristic problem solver [86]-[87], it has the problem of retaining locality or useful links because the crossover operation, the mutation operation, and the decoding of the Prüfer numbers chromosome causes big changes to the tree topology [89]. For example, in mutation, a slight change in the Prüfer number sequence representing a tree (the phenotype) may cause a big change in the tree topology (the genotype). Many useful links in the parents are then destroyed [90]. Furthermore, Prüfer numbers correspond to unconstrained spanning trees in complete graphs. When the underlying graph is not complete, the offspring produced by evolution of Prüfer numbers encoding may be invalid or disconnected. Hence, it is not suitable for feedback based routing [92] like the overlay network. Because it transmits routing information between nodes through dynamic probing and feedback of messages instead of complete knowledge of routing information of all nodes in the network. [88] proposes a graph based EA to evolve degree constrained MST but it needs to generate initial parent graphs that satisfy the degree constraint prior to evolution and order is introduced in the crossover and mutation process to produce valid children. If it happens that an edge needs to be added to repair disconnected subgraphs to form a children graph after crossover and that edge does not exist in the underlay network, no valid overlay network can be produced.

*Node Biased* encoding and *Link and Node Biased* encoding represent weights of edges or weights of nodes and edges of a network in a linear chromosome on which GA is conducted [90]. The weights in the chromosome does not necessarily equal to the weights of the edges and nodes in the network. But MST search algorithms such as PRIM and Kruskal algorithms are conducted in the network based on the set of biased weights in the

chromosome. The efficiency of search is hampered by the redundancies in this form of encoding because more than one chromosome may represent the same tree topology. Similar redundancy problem applies to *Network Random Key* encoding [141] where an arbitrary set of numbers are assigned to edges of a network. The set of numbers are sorted in ascending order in a linear chromosome. Kruskal algorithm is applied to construct MST on a network based on the order of magnitude of numbers assigned to the edges in the linear chromosome.

In [88][140], *Edge Set* encoding is used to encode edges of a network on which degree-constrained MST is evolved. The advantage of Edge Set encoding is the ability to make use of the unions and intersections of edges of the parent trees followed by edge repair or tree regeneration on them to produce offspring in the crossover process so that a high degree of heritability can be maintained in the course of evolution. But high heritability may limit the ability of the EA to explore new search space especially when the population is approaching optimal where the difference between individual trees is small.

EvoGraph generates and evolve overlay networks based on the underlay network topology with costs of flow on their links. The underlay network topology does not need to be complete. That is, each node does not need to have complete cost information of all the other nodes in the system. It can explore the use of links offered by the underlay network that lies outside the MST and SPT. The connectivity and structural information of the network can be exchanged between parents and carried to the next generation after the crossover process. The consistency of nodes can also be maintained throughout the evolutionary process.



## 9.2 Application of EvoGraph on Peer-to-Peer Overlay Network Evolution

EvoGraph start off with the generation of a population of overlay trees at random base on the underlay network for subsequent application of evolution operators. The network topologies are encoded in form of cost matrices instead of conventional encoding in linear chromosome and Prüfer number to avoid the need of post processing in decoding and the inability of retaining locality and useful links described in Section 9.1. This is the most general and direct form of encoding and EvoGraph evolves networks directly. Given an initial population of cost matrices of overlay trees, EvoGraph is able to evolve AMTCT using a set of novel evolution operators that evolve overlay network atop an underlay network and a fitness function designed to minimize TC.

Like many EAs, the EvoGraph algorithm consists of the following steps:

1. Given an underlay network, initialize a population of overlay trees at random using the connections in the underlay network.
2. Evaluate the fitness of each overlay tree.
3. Select two overlay trees for reproduction using the roulette wheel selection scheme.
4. Apply crossover and mutation operators on the selected overlay trees using connections in the underlay network.
5. Replace the least-fit overlay trees in the existing population by the newly generated offspring using the steady state reproduction scheme.
6. Repeat Steps 2 to 5 until the termination criteria are met.

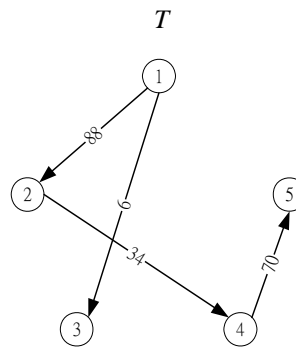
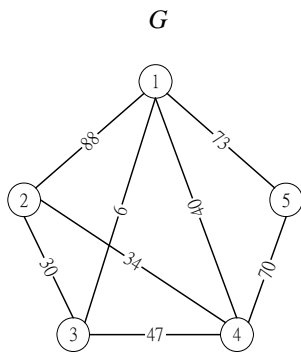
Unlike other probing based algorithms or graph based EAs, the underlay network does not need to be a complete graph for EvoGraph to operate. The crossover operator

facilitates the exchange of information between overlay trees and produce better children using the connections in the underlay tree. The mutation operator introduces variations to avoid the evolution from being trapped at local optima.

An example of encoding underlay network as an undirected graph,  $G$ , in a symmetric cost matrix,  $\mathbf{G}$ , is shown in Figure 67(a). A zero element  $c_{ij}=0$  means there is no link between node  $i$  and node  $j$ . A non-zero element in the matrix,  $c_{ij}$ , represents a link from node  $i$  to node  $j$  and  $c_{ij}$  is the cost of flow between them. A tree is considered to be a directed graph with  $|V|$  number of nodes and  $|V|-1$  number of edges starting from the origin to the leaf nodes. We encode overlay trees in cost matrices that show cost and direction between pairs of nodes and also represent adjacency between them. A non-zero element in the matrix,  $c_{ij}$ , represents a directed link from node  $i$  to node  $j$  and  $c_{ij}$  is the cost of flow between them. The directed edge is a way of encoding for the application of EvoGraph. The flow can be two way in the network. An example of overlay tree,  $T$ , encoding in a cost matrix,  $\mathbf{T}$ , is shown in Figure 67(b).

G					
	1	2	3	4	5
1	0	88	9	40	73
2	88	0	30	34	0
3	9	30	0	47	0
4	40	34	47	0	70
5	73	0	0	70	0

T					
	1	2	3	4	5
1	0	88	9	0	0
2	0	0	0	34	0
3	0	0	0	0	0
4	0	0	0	0	70
5	0	0	0	0	0



(a) Symmetric cost matrix with corresponding underlay network

(b) Overlay tree atop underlay network encoded in cost matrix as directed graph

**Figure 67:** Encoding overlay tree over an underlay network in a cost matrix

### 9.2.1 Generation of Tree on Overlay Network Atop an Underlay Network

An overlay tree spanning the source node and all the destination nodes atop the underlay network can be generated at random in form of cost matrix using the properties in Table 1. This algorithm is similar to the formalism in Section 4.1 but make use of the edges in the underlay network to generate the overlay tree. If an edge does not exist between two nodes in the underlay network, there will not be an edge between the same nodes in the overlay tree. This algorithm is also used in Section 9.2.2 and 9.2.3 to repair the degenerate trees at the intermediate stage right after the crossover of two parent trees or deletion of an edge during mutation of a tree. The idea is to identify the nodes without incoming edges and connect them to other nodes at random by having directed edges

pointed to them until one such node (without incoming edge) is left behind. The left behind node is the root of the tree. All the connections made in the overlay tree should use connections from the underlay network. The algorithm is illustrated as follow.

Both the underlay network and the overlay network are using the same set of nodes  $V = \{v_1, v_2, \dots, v_i, \dots, v_n\}$ .

Define *origin node (on)* in a tree is a node without incoming edges. It has '0' in all its column cells in the cost matrix.

Let  $G(V, E)$  be the graph of the underlay network with the set of nodes  $V = \{v_1, v_2, \dots, v_n\}$  and set of edges  $E = \{e(v_i, v_j) : v_i, v_j \in V\}$  where  $e(v_i, v_j)$  is a directed edge from  $v_i$  to  $v_j$ .

$\mathbf{G}$  be the cost matrix of  $G(V, E)$ .

$c_{v_i v_j}$  be the communication cost between node  $v_i$  and  $v_j$  in  $\mathbf{G}$  and  $G(V, E)$

$T(V', E')$  be a tree in the underlay network that contains the source node  $v_s$  with  $V' \subset V, E' \subset E$ .

$\mathbf{T}$  be the cost matrix of  $T(V', E')$ .

$C_{v_i v_j}$  be the communication cost between node  $v_i$  and  $v_j$  in  $\mathbf{T}$  and  $T(V', E')$ .

$|X|$  notates the number of elements in set  $X$ .

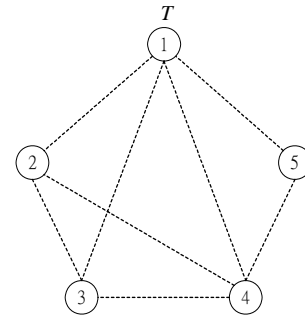
1. initialize  $\mathbf{T}$  with  $C_{v_i v_j} = 0$  for all  $C_{v_i v_j} \in \mathbf{T}$ ;
2.  $W \leftarrow$  all *ons* in  $\mathbf{T}$ ; (\* find all origin nodes in  $\mathbf{T}$  to form a set  $W$  \*)
3.  $V' \leftarrow \{v_s\}, E' \leftarrow \emptyset$ ; (\* assign source node  $v_s$  as the first node in the tree  $T^*$  \*)
4. while  $|W| > 1$  (\* continue the while loop until only one *on* is left behind \*)
5. find a set of nodes  $B \subset V'$  in  $T$  that have connections to nodes outside  $T$  in the underlay network (i.e. connected to nodes in  $V \setminus V'$ );
6. select at random a node  $v_r \in B$ ;
7. select at random some nodes  $v_i \in V \setminus V'$  that connect to  $v_r$  in the underlay

- network;
8. for all  $v_i$ ,  $C_{v_r, v_i} \leftarrow c_{v_r, v_i}$ ,  $v_i \cup V'$ ,  $e(v_r, v_i) \cup E'$ ; (\*tree  $T(V', E')$  construction by assigning cost  $c_{v_r, v_i}$  in  $\mathbf{G}$  to  $C_{v_r, v_i}$  in  $\mathbf{T}$ , include node  $v_i$  in  $V'$  and edge  $e(v_r, v_i)$  in  $E'$  \*)
  9. count  $|W|$ ; (\* count the number of *ons* in  $\mathbf{T}$ \*)
  10. end

An example on generation of an overlay tree from the underlay network in Figure 67(a) is illustrated in Figure 68. Node '1' is the source node  $v_s$ .

1. Initialize network cost matrix by setting  $c_{ij} = 0$  for all nodes in  $T$ ;
2.  $W = V = \{ '1', '2', '3', '4', '5' \}$ ;
3.  $V' = \{ '1' \}$ ,  $E' = \phi$
4. while  $|W|/5 > 1$

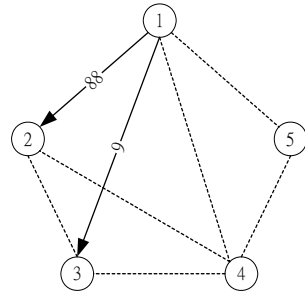
T					
	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0



Iteration 1:

5.  $B = \{ '1' \}$ ;
6.  $\{ '2', '3', '4', '5' \} \in V \setminus V'$  connected to node in  $B$ ;
7. select from  $B$   $v_r = '1'$ ,  $\{ '2', '3', '4', '5' \} \in V \setminus V'$  is connected to  $v_r = '1'$  in the communication network;
8. select at random  $'2', '3'$  from  $V \setminus V'$
9. connect  $v_r$  to nodes  $'2', '3'$   
 $C_{12} \leftarrow (c_{12} = 88)$ ,  $C_{13} \leftarrow (c_{13} = 9)$ ,  
 $V' = '2' \cup ('3' \cup \{ '1' \})$ ,  
 $E' = e(1,2) \cup (e(1,3) \cup \phi)$ ;
10.  $W = \{ '1', '4', '5' \}$ ;
11.  $|W|/5 = 3 > 1$ ;

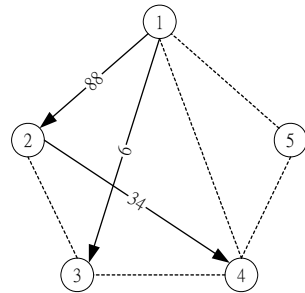
	1	2	3	4	5
1	0	88	9	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0



Iteration 2:

12.  $B = \{ '1', '2', '3' \}$ ;
13.  $\{ '4', '5' \} \in V \setminus V'$  connected to the nodes in  $B$ ;
14. select at random from  $B$   $v_r = '2'$ ,  $\{ '4', '5' \} \in V \setminus V'$  is connected to  $v_r = '2'$  in the communication network;
15. select  $'4'$  from  $V \setminus V'$ ;
16. connect  $v_r$  to node  $'4'$   $C_{24} \leftarrow (c_{24} = 34)$ ,  
 $V' = '4' \cup ('1', '2', '3')$ ,  
 $E' = e(2,4) \cup \{ e(1,2), e(1,3) \}$ ;
17.  $W = \{ '1', '5' \}$ ;
18.  $|W|/5 = 2 > 1$ ;

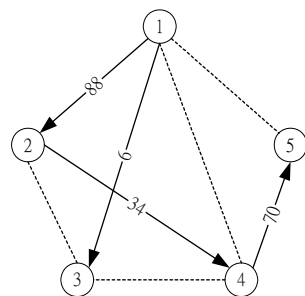
	1	2	3	4	5
1	0	88	9	0	0
2	0	0	0	34	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0



Iteration 3:

19.  $B = \{ '1', '4' \}$ ;
20.  $\{ '5' \} \in V \setminus V'$  connected to the nodes in  $B$ ;
21. select at random from  $B$   $v_r = '4'$ ,  $\{ '5' \} \in V \setminus V'$  is connected to  $v_r = '4'$ ;
22. select  $'5'$  from  $V \setminus V'$ ;
23. connect  $v_r$  to node  $'5'$   
 $C_{45} \leftarrow (c_{45} = 70)$ ,  
 $V' = '5' \cup ('1', '2', '3', '4')$ ,  
 $E' = e(4,5) \cup \{ e(1,2), e(1,3), e(2,4) \}$ ;
24.  $W = \{ '1' \}$ ;
25.  $|W|/5 = 1$ ;
26. end

	1	2	3	4	5
1	0	88	9	0	0
2	0	0	0	34	0
3	0	0	0	0	0
4	0	0	0	0	70
5	0	0	0	0	0

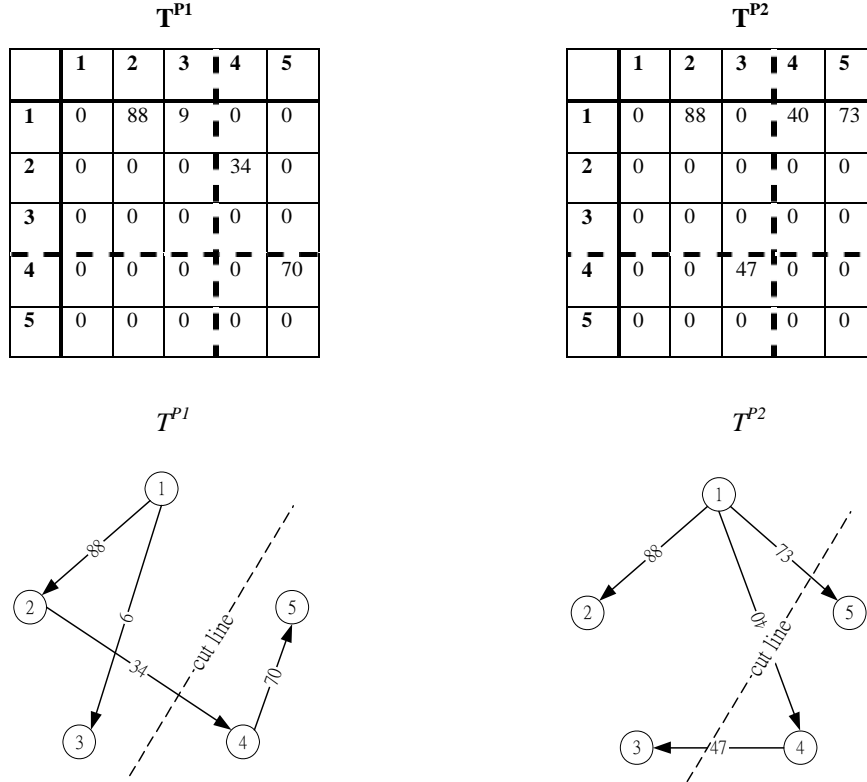


**Figure 68:** An overlay tree generated atop an underlay network

## 9.2.2 Crossover of Networks

The *random crossover* is applied to crossover of two overlay trees. This is similar to the formalism in Section 4.2.2 but it uses the edges in the underlay network for reconnection after breaking up the trees in the crossover stage. Given two overlay trees,  $T^{P1}(V^{P1}, E^{P1})$  and  $T^{P2}(V^{P2}, E^{P2})$  with corresponding cost matrices  $\mathbf{T}^{P1}$  and  $\mathbf{T}^{P2}$ , the operation can be described as follows. For each of  $\mathbf{T}^{P1}$  and  $\mathbf{T}^{P2}$ , a crossover point between two neighboring rows and columns in a cost matrix is selected randomly. One submatrix from each of the split parent matrix is then swapped to form two new child matrices. Connections within the submatrices are retained throughout the process while connections outside them are deleted. New edges are generated with repairing in each of the resulting matrices to form two children,  $\mathbf{T}^{C1}$  and  $\mathbf{T}^{C2}$ . The formalism is illustrated with an example below.

1. Given an underlay network, two overlay tree cost matrices  $\mathbf{T}^{P1}$  and  $\mathbf{T}^{P2}$  are constructed with corresponds to overlay tree topologies  $T^{P1}(V^{P1}, E^{P1})$  and  $T^{P2}(V^{P2}, E^{P2})$ , with node sets  $\{v_1^{P1}, v_2^{P1}, \dots, v_i^{P1}, v_{i+1}^{P1}, \dots, v_n^{P1}\}$  and  $\{v_1^{P2}, v_2^{P2}, \dots, v_i^{P2}, v_{i+1}^{P2}, \dots, v_n^{P2}\}$  respectively. An example using the underlay network in Figure 67(a) is shown in Figure 69. In the example number of nodes of both trees is  $n=5$  and node indices of both trees are  $v_i^{P1} = v_i^{P2} = 'i'$ .
2. A crossover point in each of  $\mathbf{T}^{P1}$  and  $\mathbf{T}^{P2}$  is randomly selected.

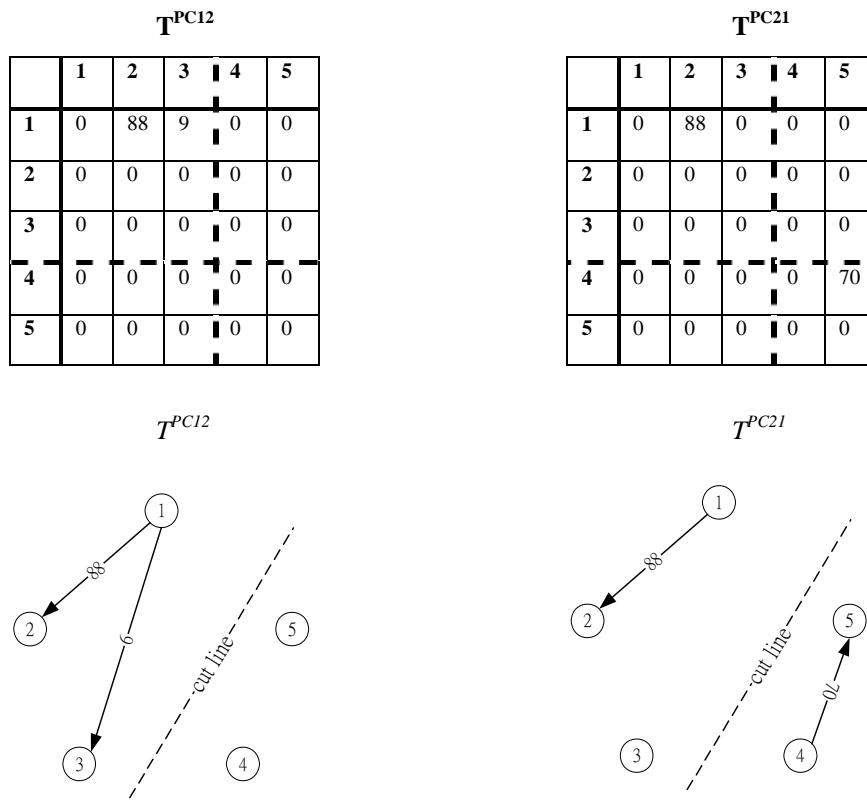


(a) **Figure 69:** Parent trees  $T^{P1}$  and  $T^{P2}$

(b)

- Assume that the crossover point for  $T^{P1}$  is between  $v_i^{P1}$  and  $v_{i+1}^{P1}$  and for  $T^{P2}$  is between  $v_i^{P2}$  and  $v_{i+1}^{P2}$ , the lower right portions of these two cost matrices are then swapped so that the rows and columns corresponding to  $\{v_{i+1}^{P1}, \dots, v_n^{P1}\}$  are swapped with the rows and columns corresponding to  $\{v_{i+1}^{P2}, \dots, v_n^{P2}\}$  to form two matrices  $T^{PC12}$  and  $T^{PC21}$ . The valid node labels for  $T^{PC12}$  are therefore given by  $\{v_1^{P1}, v_2^{P1}, \dots, v_i^{P1}, v_{i+1}^{P2}, \dots, v_n^{P2}\}$  and for  $T^{PC21}$  by  $\{v_1^{P2}, v_2^{P2}, \dots, v_i^{P2}, v_{i+1}^{P1}, \dots, v_n^{P1}\}$ . In the example,  $i=3$ .
- All cell entries in each of  $T^{PC12}$  and  $T^{PC21}$  are scanned to remove invalid edges. Degenerate trees  $T^{PC12}$  and  $T^{PC21}$  are formed.





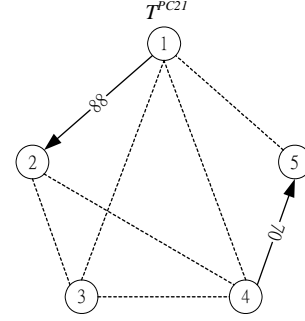
(a) (b)  
**Figure 70:** Swap nodes '4', '5' between  $T^{P1}$  and  $T^{P2}$  on the right hand side of the cut line and delete invalid edges.

5. Regenerate a spanning tree from each of the degenerate trees using similar algorithm on spanning tree generation at random as follow.
  1. find  $T$  (the tree containing the source node  $v_s$ ) in the degenerate tree.
  2. repeat the 'while loop' from step 4 to step 10 in the algorithm on generating spanning tree at random in Section 9.2.1 with the following modifications (i) step 7 modified to select at random 'one' node  $v_i \in V \setminus V'$  instead of 'some' nodes to connect to  $v_r$  and (ii) step 8 modified to also include into  $T$  the subtree from the other parent induced by the new connection  $e(v_r, v_i)$ , if any, then followed by re-alignment of edge directions in  $T$ . Note that the induced subtree from the other parent should not contain the source node according to our way of selection nodes to be swapped in the crossover process.

An example using the degenerate tree  $\mathbf{T}^{\text{PC}21}$  in Figure 70(b) to generate children  $\mathbf{T}^{\text{C}2}$  is illustrated in Figure 71. Similarly, the other children  $\mathbf{T}^{\text{C}1}$  is generated from  $\mathbf{T}^{\text{PC}12}$  by the same algorithm. The crossover process is then completed. It can be observed in the example that  $e(2,3)$  is not included in the parents  $T^{P1}$  and  $T^{P2}$  but in the underlay network  $G$ . Hence the crossover may use edges outside the parents. In other words, the crossover of MST and SPT of an underlay network may produce children containing edges outside them.

1. find  $T(V',E')$  in  $T^{PC21}$ .  
 $V'=\{ '1', '2' \}$ ,  $E'=\{ e(1,2) \}$ ;
2.  $W=\{ '1', '3', '4' \}$ ;
3. while  $|W|=3>1$

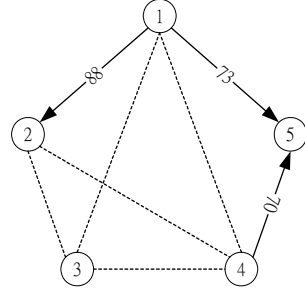
	$T^{PC21}$				
	1	2	3	4	5
1	0	88	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	70
5	0	0	0	0	0



iteration 1

4.  $B = \{ '1', '2' \}$ ;
5.  $\{ '3', '4', '5' \} \in V \setminus V'$   
connected to the nodes in  $B$ ;
6. select at random from  $B$   
 $v_r = '1'$ ,  $\{ '3', '4', '5' \} \in V \setminus V'$   
is connected to  $v_r = '1'$  in the  
underlay network;
7. select at random '5' from  
 $V \setminus V'$ ;
8. connect  $v_r$  to node '5'  
 $C_{15} \leftarrow (c_{15}=73)$ ,  
 $V' = '4' \cup ('5' \cup \{ '1', '2' \})$ ,

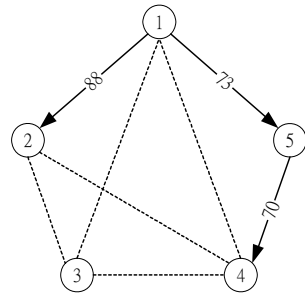
	1	2	3	4	5
1	0	88	0	0	73
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	70
5	0	0	0	0	0



$E' = e(4,5) \cup (e(1,5) \cup e(1,2))$ ;  
it can be detected that column '5' in  
 $T^{PC12}$  has more than one non-zero cell.  
This violates the tree property in Table  
1.

9. re-align direction of edges in  $T$   
from the source node toward the  
leaf nodes by reversing direction  
of  $e(4,5)$  to  $e(5,4)$ .  
 $V' = \{ '1', '2', '4', '5' \}$ ,  
 $E' = \{ e(1,5), e(1,2), e(5,4) \}$ ;
10.  $W = \{ '1', '3' \}$ ;
11.  $|W|=2>1$

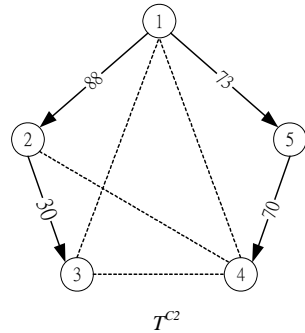
	1	2	3	4	5
1	0	88	0	0	73
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	70	0



Iteration 2:

12.  $B = \{ '1', '2', '4' \}$ ;
  13.  $\{ '3' \} \in V \setminus V'$  connected to  
nodes in  $B$ ;
  14. select at random  $v_r = '2'$ ,  
 $\{ '3' \} \in V \setminus V'$  is connected to  
 $v_r = '2'$  in the underlay network;
  15. select '3' from  $V \setminus V'$ ;
  16. connect  $v_r$  to node '3'  
 $C_{23} \leftarrow (c_{23}=30)$ ,  
 $V' = '3' \cup \{ '1', '2', '4', '5' \}$ ,
- $E' = e(2,3) \cup \{ e(1,2), e(1,5), e(5,4) \}$ ;
17.  $W = \{ '1' \}$ ;
  18.  $|W|=1$ ;
  19. end

	1	2	3	4	5
1	0	88	0	0	73
2	0	0	30	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	70	0



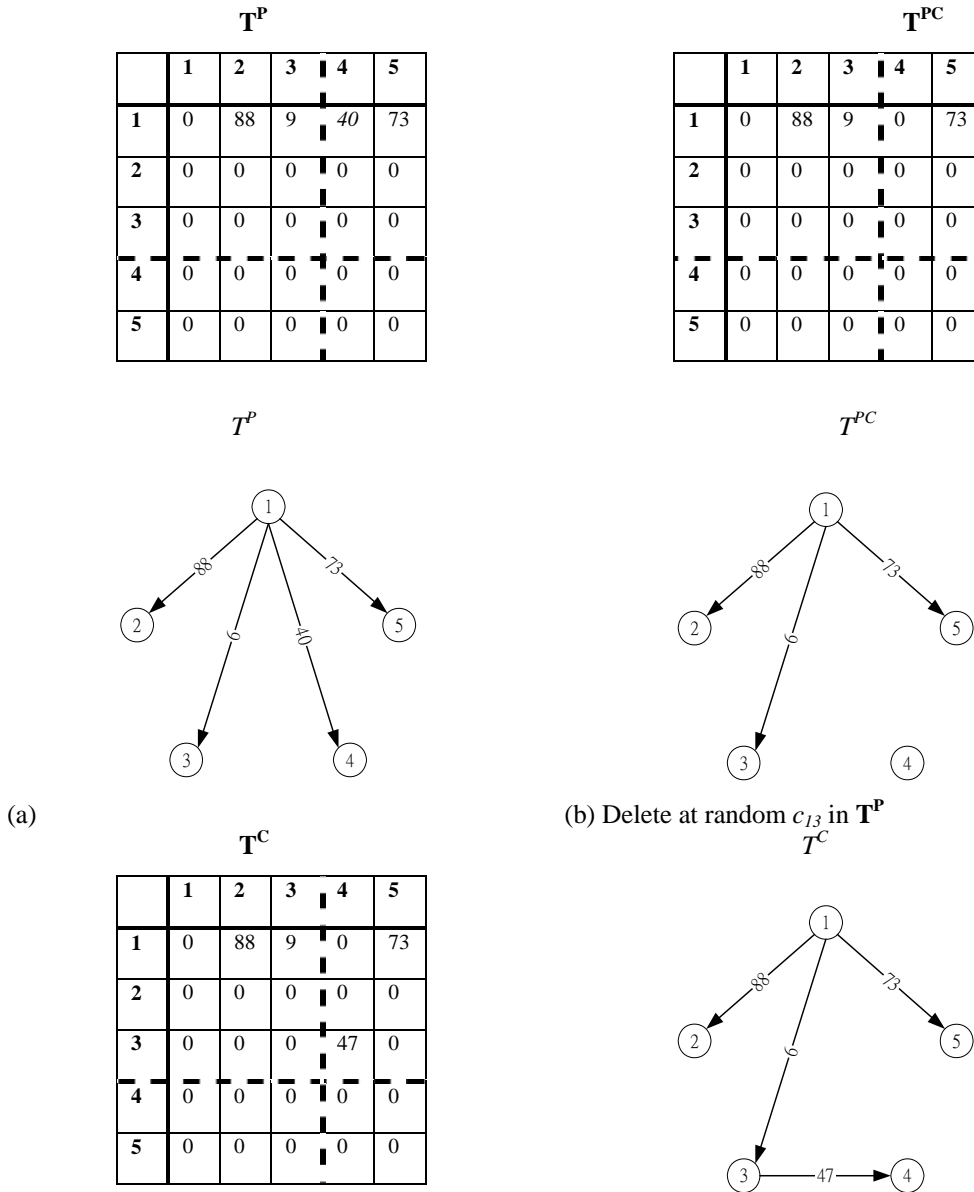
**Figure 71:** Generation of children spanning tree  $T^{C1}$  from degenerate tree  $T^{PC21}$  after exchange of subtrees in crossover

### 9.2.3 Mutation of Networks

Mutation can be very useful when a population is to avoid being trapped in a suboptimal state. Because of the large search space in network evolution, mutation plays an especially important role in network evolution. To mutate networks to avoid getting trapped at local minima. The *Number-of-Edge* mutation operator allows us to increase or decrease the number of edges in a network by one. The formalism is similar to Section 4.3.1 but it uses the edges in the underlay network to for addition or deletion of an edge at random. The details are given below.

1. For an overlay tree  $T^P(V^P, E^P)$  with edge set,  $E^P$ , we construct its corresponding cost matrix as  $\mathbf{T}^P$ .
2. Select a 'non-zero' cell in  $\mathbf{T}^P$ . Convert the selected cell to '0' to form an intermediate network  $T^{PC}$  and its corresponding cost matrix  $\mathbf{T}^{PC}$ . Use the overlay tree construction algorithm step 2 to 15 in Section 9.2.1 to form a new connection to generate a child overlay tree cost matrix  $\mathbf{T}^C$ .

An example of the *Number-of-Edge* mutation is given in Figure 72 below.



(c) Add  $c_{78}$  according to overlay tree generation algorithm in Section 9.2.1.

**Figure 72:** Network mutation demonstrated

### 9.2.4 Fitness Function

The purpose of the fitness function is to minimize TC, which equals TPC+TEC. TC is proportional to the number of nodes in the overlay tree,  $|V|$ , and the costs of links between the nodes,  $c_{ij}$ , which is generated at random not large than 100 in the underlay network

prior to the experiments. The fitness function to be maximized in the evolutionary process is devised as follow.

$$fitness = \frac{100 \times |V|}{TC} = \frac{100 \times |V|}{TPC + STC}$$

The numerator is a constant with respect to the number of nodes in the underlay network. The number 100 is a constant to adjust the value of the numerator in the order of the denominator. The denominator  $TC = TPC + TEC$  is the value to be minimized.

### 9.3 Experiments

The experiments starts at an 8 nodes underlay network generated at random. The following underlay networks have 2 nodes added at each interval. The step up continues to 24 nodes. The graphs of these underlay networks and their cost matrices are shown in Appendix 6. The source node is the upper left most node ‘1’ in the cost matrix. The SPT and MST are constructed by Dijkstra’s algorithm and PRIM algorithm respectively for each of the underlay network. Their  $TC_{SPT}$  and  $TC_{MST}$  are calculated for the termination criteria of evolution. The degree of nodes in the random underlay network has an average between 3.4 and 6.6667 shown in Table 24. The degree density is obtained by dividing the total degree of all nodes in a network by the number of nodes. For each of the underlay network generated, the cost of flow  $c_{ij}$  between a pair of nodes indexed ‘ $i$ ’, ‘ $j$ ’ is an integer generated randomly at a limit of 100.

V	Min degree	Max degree	Degree density
8	4	6	5.25
10	4	6	5.6
12	4	6	5.6667
14	4	6	5.4286
16	5	6	5.75
18	4	6	5.6667
20	2	8	5.5
22	2	8	4.4545
24	2	8	6
<b>Average</b>	3.4	6.6667	5.48

**Table 24:** Degree of randomly generated underlay network

### 9.3.1 Initialization and Evolution Parameters

Fifty overlay tree cost matrices are generated at random for each underlay network as the initial population. For EvoGraph, the relative proportion of mutation to crossover is higher in comparing with conventional GA to increase the role of random search. Two overlay trees are then selected for crossover. As there is a background underlay network which is not a complete graph, the search space is smaller and less number of mutations is used to explore the search space. The numbers of random crossover applied and *Number-of-Edge* mutation applied are fixed at 10 and 1 respectively. Steady State Reproduction and roulette wheel selection is adopted as stated in Section 9.2. There can be more than one AMTCT satisfying the  $TC < \min(TC_{SPT}, TC_{MST})$  condition. Though we do not need to obtain the absolute minimum TC, the search is extended for another 100 generations to seek further improvement after the first AMTCT is found. The termination criteria are

1. AMTCT is found. That is,  $TC < \min(TC_{SPT}, TC_{MST})$ , and
2. there is no improvement in fitness for 100 generations after the first AMTCT is found.

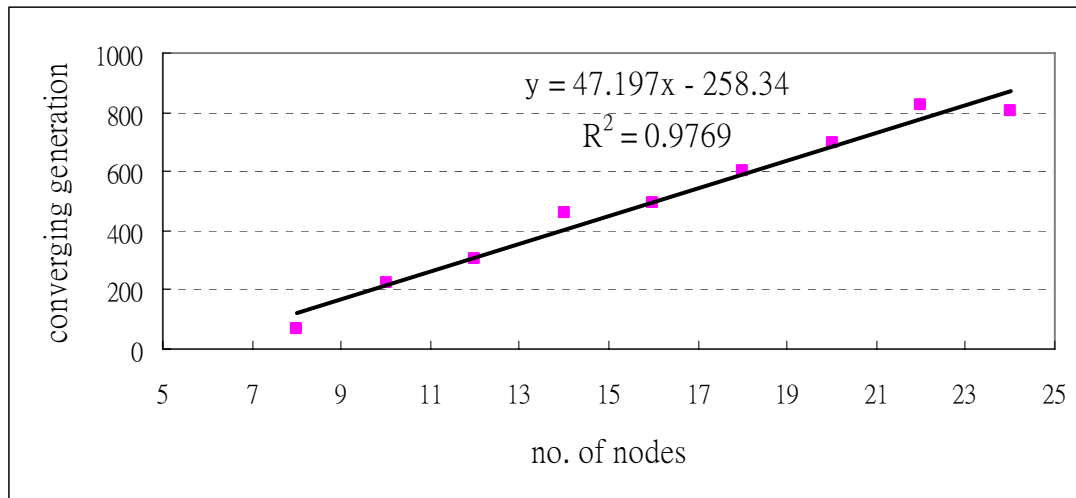
### 9.3.2 Experimental Results

Ten experiments are carried out for each of the underlay network. The average number of generations of convergence is obtained for each underlay network to find out the trend of convergence in relation to the number of nodes. The results with mean, spread, standard deviation, and the spread-standard deviation ratio on number of generations to reach convergence are summarized in Table 25. It can be observed that the spread-standard deviation ratio on the converging generation is in the 0.3 to 0.4 range. The relation between the number of nodes and the converging generation is linear with least square regression at 0.9769 in as shown in Figure 73.

Exp	V =8	V =10	V =12	V =14	V =16	V =18	V =20	V =22	V =24
1	2	297	322	615	393	569	838	424	882
2	64	174	335	458	393	683	575	1163	1212
3	67	227	338	611	512	270	766	1184	577
4	47	270	279	408	658	532	393	673	985
5	43	143	345	282	613	703	735	1097	1163
6	192	221	262	288	363	515	725	495	644
7	8	287	424	403	521	835	858	986	1058
8	65	150	307	610	359	585	505	835	617
9	10	307	342	552	422	933	727	458	334
10	167	177	116	349	671	422	804	933	587
mean	66.5	225.3	307	457.6	490.5	604.7	692.6	821.2	805.9
s.d.	64.61	62.41	80.01	132.19	122.15	193.51	152.43	297.09	294.16
spread	190	164	308	333	312	663	465	760	878
s.d./spread	0.34	0.38	0.26	0.40	0.39	0.29	0.33	0.39	0.34

**Table 25:** Converging generation for experiments on different number of nodes

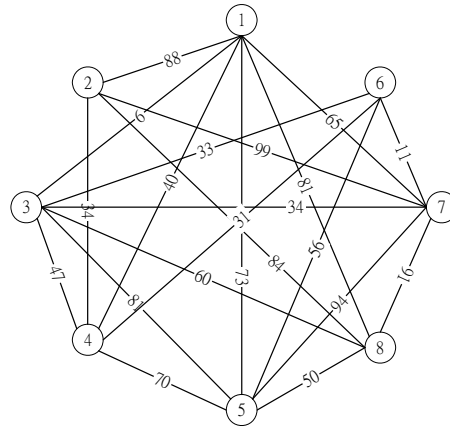




**Figure 73:** Linear relation between number of nodes and converging generations on search of AMTCTs by EvoGraph

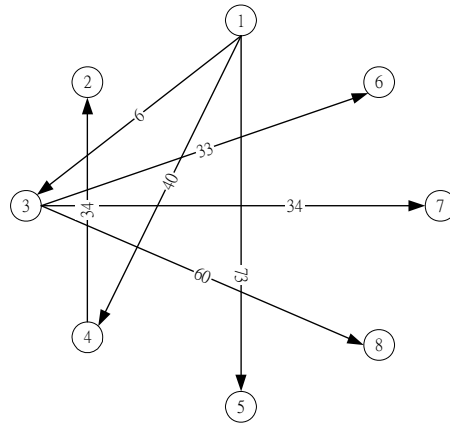
The AMTCT with the lowest TC for the 10 experiments for each of the underlay network with  $|V|$  number of nodes are shown in Appendix 6. The edges of the AMTCTs with the lowest TCs are highlighted by bold numbers and underlined in respective cost matrices. We use the 5<sup>th</sup> experiment on underlay network with  $|V|=8$  as an example. The underlay network is shown in Figure 74(a), the SPT of the underlay network is shown in Figure 744(b) with the tree edges highlighted in bold numbers and underline in the cost matrix, its MST is shown in Figure 74(c), and its AMTCT searched by EvoGraph is shown in Figure 74(d). The graph of fitness vs the number of generations of evolution is shown in Figure 74(e). EvoGraph converges at the 43<sup>rd</sup> generation. The corresponding cost savings on TC comparing with MST and SPT is illustrated on Figure 74(f).

	1	2	3	4	5	6	7	8
1	0	88	9	40	73	0	65	81
2	88	0	0	34	0	0	99	84
3	9	0	0	47	81	33	34	60
4	40	34	47	0	70	31	0	0
5	73	0	81	70	0	56	94	50
6	0	0	33	31	56	0	11	0
7	65	99	34	0	94	11	0	91
8	81	84	60	0	50	0	91	0



(a) Cost matrix and underlay network of  $|V|=8$ .

	1	2	3	4	5	6	7	8
1	0	88	<u>9</u>	<u>40</u>	<u>73</u>	0	65	81
2	88	0	0	34	0	0	99	84
3	9	0	0	47	81	<u>33</u>	<u>34</u>	<u>60</u>
4	40	<u>34</u>	47	0	70	31	0	0
5	73	0	81	70	0	56	94	50
6	0	0	33	31	56	0	11	0
7	65	99	34	0	94	11	$\infty$	91
8	81	84	60	0	50	0	91	0

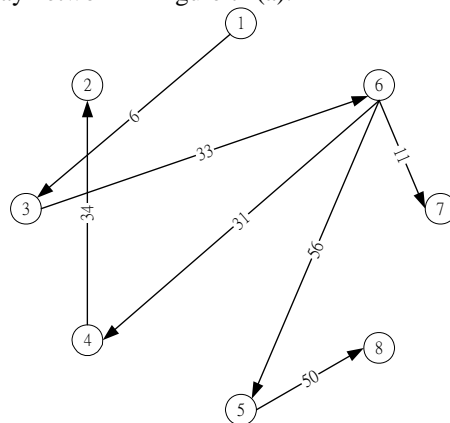


$$TPC_{SPT} = 9 + 40 + 73 + 42 + 43 + 69 + 74 = 350$$

$$TEC_{SPT} = 9 + 40 + 73 + 33 + 34 + 60 + 34 = 283$$

(b) Cost matrix and corresponding SPT over the underlay network in Figure 74(a).

	1	2	3	4	5	6	7	8
1	0	88	<u>9</u>	40	73	0	65	81
2	88	0	0	34	0	0	99	84
3	9	0	0	47	81	<u>33</u>	34	60
4	40	<u>34</u>	47	0	70	31	0	0
5	73	0	81	70	0	56	94	<u>50</u>
6	0	0	33	<u>31</u>	<u>56</u>	0	<u>11</u>	0
7	65	99	34	0	94	11	0	91
8	81	84	60	0	50	0	91	0

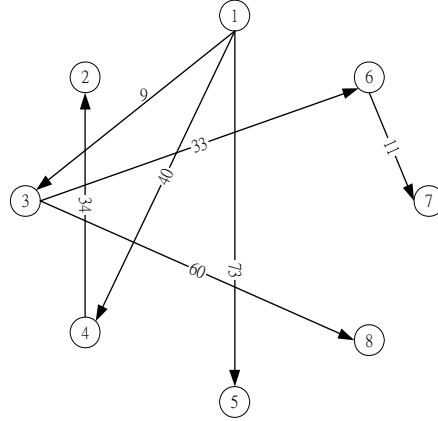


$$TPC_{MST} = 9 + 107 + 73 + 98 + 42 + 53 + 148 = 530$$

$$TEC_{MST} = 9 + 33 + 31 + 56 + 34 + 50 + 11 = 224$$

(b) Cost matrix and corresponding MST over the underlay network in Figure 74(a).

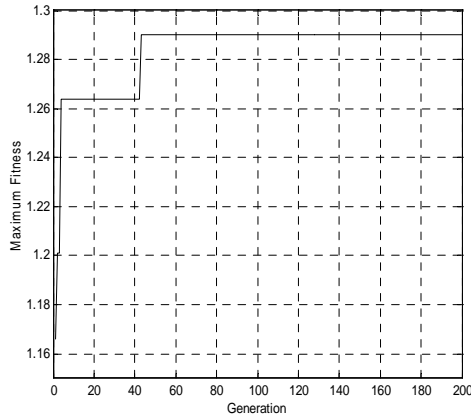
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>
<b>1</b>	0	88	<u>9</u>	<u>40</u>	<u>73</u>	0	65	81
<b>2</b>	88	0	0	34	0	0	99	84
<b>3</b>	9	0	0	47	81	<u>33</u>	34	<u>60</u>
<b>4</b>	40	<u>34</u>	47	0	70	31	0	0
<b>5</b>	73	0	81	70	0	56	94	50
<b>6</b>	0	0	33	31	56	0	<u>11</u>	0
<b>7</b>	65	99	34	0	94	11	0	91
<b>8</b>	81	84	60	0	50	0	91	0



$$TPC_{AMTCT} = 74 + 9 + 40 + 73 + 42 + 53 + 69 = 360$$

$$TEC_{AMTCT} = 9 + 33 + 34 + 73 + 40 + 60 + 11 = 260$$

(d) Cost matrix and corresponding AMTCT of underlay network in Figure 74(a). AMTCT highlighted in bold and underlined numbers in cost matrix.



	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
SPT	350	283	633
<b>AMTCT</b>	<b>360</b>	<b>260</b>	<b>620</b>
net	-10	23	13
% saving			2%

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
MST	530	224	754
<b>AMTCT</b>	<b>360</b>	<b>260</b>	<b>620</b>
net	170	-36	134
% saving			18%

(e) Fitness vs number of generations

(f) TC saving comparison

**Figure 74:** AMTCT searched by EvoGraph atop underlay network in Figure 74(a) and cost comparison with corresponding SPT, MST.

The net TC savings of the fittest AMTCTs for the nine underlay networks ( $|V|=8$  to  $|V|=24$  at 2 nodes interval) are listed in Appendix 6. The percentage of TC savings for each underlay network with respect to  $TC_{SPT}$  and  $TC_{MST}$  are summarized in Table 26. The amount of cost saving depend on the underlay network topology and the cost of flows in the network. This varies from case to case. Generally total cost savings on MST is higher than SPT.

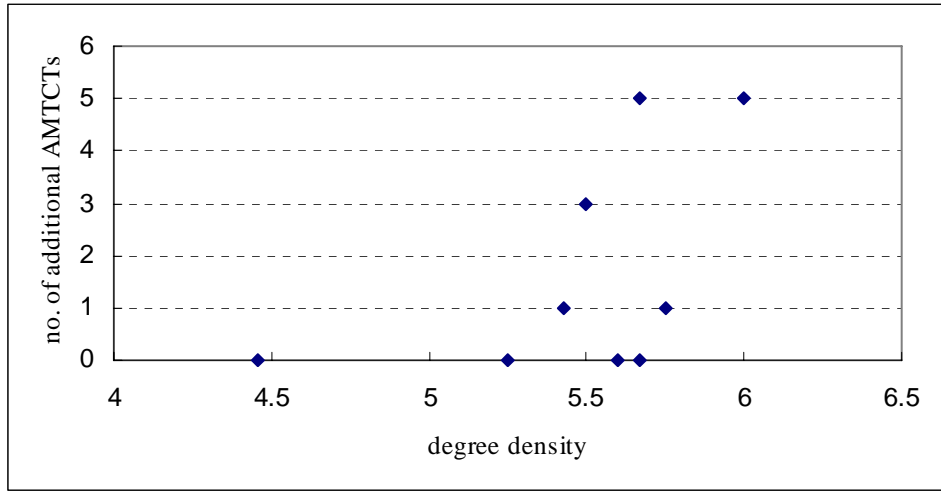
IVI	% TC saving on TC <sub>SPT</sub>	% TC saving on TC <sub>MST</sub>
8	2	18
10	3	13
12	5	3
14	3	8
16	2	6
18	4	8
20	10	21
22	1	3
24	1	17
average	4	11

**Table 26:** Percentage cost saving results of AMTCTs on TC<sub>SPT</sub> and TC<sub>MST</sub> on each underlay network

The fitnesses of all experiments are shown in Table 27. There is at least one AMTCT discovered for each underlay network. It can be observed that as the number of nodes in the system increases, the number of AMTCTs discovered increases. The fitnesses of the additional AMTCTs are highlighted in *italics* in Table 27. Their fitnesses are below the fittest AMTCT but they are acceptable because they satisfy the condition  $TC < \min(TC_{SPT}, TC_{MST})$ . There is a weak positive correlation (correlation coefficient = 0.51) between the degree density of the underlay network and the number of additional AMTCTs. The trend is shown in Figure 75. This may be due to the fact that more choices are offered by the high degree density underlay networks for path selection.

Exp	V =8	V =10	V =12	V =14	V =16	V =18	V =20	V =22	V =24
1	1.2903	1.3333	1.4742	1.7677	1.4222	<u>1.7408</u>	1.0582	1.2702	1.4371
2	1.2903	1.3333	1.4742	1.7677	1.4222	<u>1.7408</u>	1.0582	1.2702	1.4371
3	1.2903	1.3333	1.4724	1.7677	<u>1.3998</u>	1.7630	<u>1.0417</u>	1.2702	<u>1.4286</u>
4	1.2903	1.3333	1.4742	1.7677	1.4222	<u>1.7341</u>	1.0582	1.2702	<u>1.4286</u>
5	1.2903	1.3333	1.4742	1.7677	1.4222	1.7630	1.0582	1.2702	1.4371
6	1.2903	1.3333	1.4742	1.7677	1.4222	1.7630	<u>1.0444</u>	1.2702	<u>1.4286</u>
7	1.2903	1.3333	1.4742	1.7677	1.4222	1.7630	1.0582	1.2702	<u>1.4286</u>
8	1.2903	1.3333	1.4742	<u>1.7588</u>	1.4222	<u>1.7176</u>	1.0582	1.2702	1.4371
9	1.2903	1.3333	1.4724	1.7677	1.4222	<u>1.7391</u>	1.0582	1.2702	<u>1.4286</u>
10	1.2903	1.3333	1.4724	1.7677	1.4222	1.7630	<u>1.0493</u>	1.2702	1.4371
Highest AMTCT fitness	1.2903	1.3333	1.4742	1.7677	1.4222	1.7630	1.0582	1.2702	1.4371
No. of AMTCTs with fitness below the fittest AMTCT	0	0	0	1	1	5	3	0	5
Degree density	5.25	5.6	5.6667	5.4286	5.75	5.6667	5.5	4.4545	6

**Table 27:** Fitness of AMTCTs generated in relation to degree density. The fitness of AMTCTs below the maximum fitness is underlined



Correlation coefficient = 0.51.

**Figure 75:** Relation between degree density and number of additional AMTCTs.

## Research Conclusion and Future Directions

---

### 10.1 Research Conclusion and Contribution

The objective of this thesis is to create a general evolutionary algorithm for graphs stated in Section 1.2. As discussed in Chapter 2, the current researches on graph evolution mainly focus on ANN evolution, tree evolution, and a few application specific networks. These evolutionary algorithms are designed to evolve specific graph topologies and they cannot be easily adapted to evolve graphs without specific topologies. In this thesis, EvoGraph is developed to evolve graphs without specific topologies by encoding them in adjacency matrices. Evolution operators on crossovers and mutations are designed to work on these encoded adjacency matrices. EvoGraph contributes to both the theory and application of graph evolution.

On the theoretical aspect, it overcomes the limitation on encoding graphs in conventional linear chromosomes which requires the length of chromosomes to be in square numbers or some other form of indirect encoding in order to present the connection status between all nodes. This length limitation imposes unnecessary constraints on crossovers and mutations. EvoGraph is applied to encode the most common types of graph topologies including tree and ANN. New crossover operators comparable to existing GP and ANN evolution are developed. The step by step comparison between EvoGraph operation and standard GP as well as conventional ANN evolution operation in Chapter 4 is to demonstrate that one operation of EvoGraph *random crossover* may produce the same effect as several conventional evolution

operations in standard GP and ANN. This random breakdown and recombination of topologies in *random crossover* may also be a potential cure to the problem of bloat in standard GP. Experiments in Chapter 4 provide some initial evidence on EvoGraph as a more efficient evolutionary algorithm when compared with standard evolution operators in EP and GP on evolution of graphs without specific topologies and tree.

On the application aspect, EvoGraph successfully implement designs that are conventionally considered as esoteric processes monopolized by artist or architects. This thesis applies evolutionary algorithms to art creation through the experiments designed for EvoGraph. In Chapter 5, it is demonstrated that EvoGraph could be an invaluable tool in that it can help architects convert the descriptive requirements of the client to several possible architectural space topologies that satisfy the client's needs. This saves the architect from doing spatial configuration manually by the conventional approach through trial-and-error. In Chapter 6, the hybrid evolutionary algorithm with EvoGraph and GA solves the puzzle that has been bordering architects and graphics designers for ages. That is, how to create a regular geometry with given number of nodes and edges of a base frame module? We can now evolve regular space frame modules or regular three-dimensional geometric figures with small number of generations for convergence. The same algorithm can be adopted for two-dimensional graphics design. Experiments in Chapter 7 use EvoGraph for visual art creation and successfully create Mondrian paintings according to the artist's original aesthetic intention. In Chapter 8, hydrogen depleted graphs of molecules are encoded in EvoGraph to evolve drugs by using the basic chemical properties such as atomic weight and valance of an atom in a molecule. In Chapter 9, EvoGraph is adapted to evolve peer-to-peer overlay network atop an underlay

network. It solves the problem in the existing graph algorithms on not having a clear criterion of trade off between MST and SPT on network optimization. EvoGraph does not need to work on a complete cost graph of the underlay network and it also create an opportunity of using appropriate links in the underlay network outside the MST and SPT.

## **10.2 Limitations of the Research**

This thesis has developed the fundamental principles of graph evolution. Like most of other evolutionary algorithms [1][3], it starts off on a limited scale for the clarity of illustration of concepts. EvoGraph faces the same problems encountered by other heuristic search methods such as there is no guarantee of convergence at the universal optimum, difficulty in selecting value of parameters for operators and consistencies of results. However, there are other limitations due to the properties of graphs.

Whilst graph has been notorious on the number of topological combinations it can generate with relatively small number of nodes. For example, given 10 nodes, there are more than ten million graph topologies. The number of graph topologies increase exponentially with the number of nodes. This figure is yet to include the different combinations bring about by permutation of nodes. Fortunately, with the other constraints imposed in the search experiments, EvoGraph is able to approach convergence at manageable number of generations. To further improve the efficiency, experiments on different combinations on evolution parameters can be conducted.

## **10.3 Future Directions of Research**

This thesis has developed the general algorithm for graph evolution as an entry portal that leads to a vast landscape of researches ahead. According to the limitations outlined in

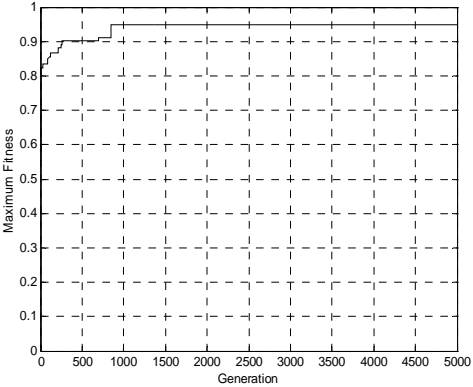
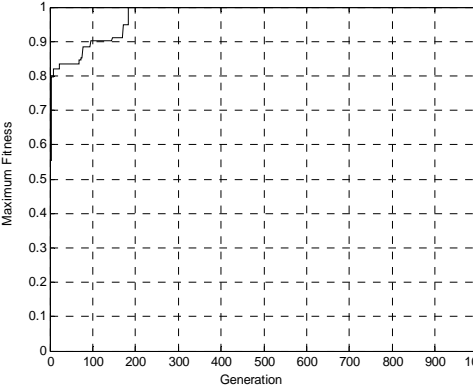
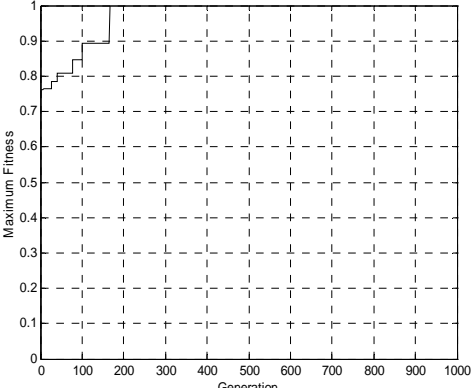
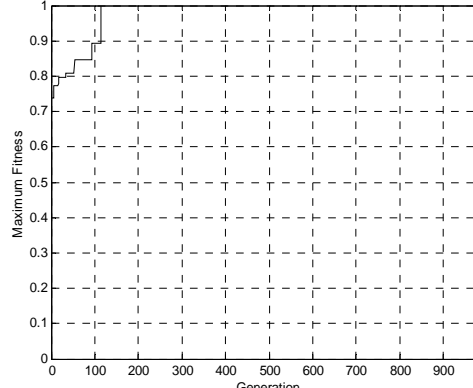


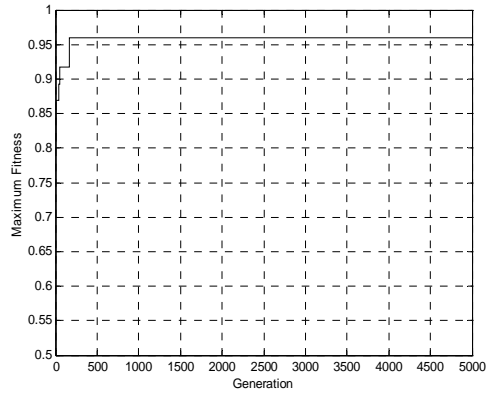
Section 10.2, the problem of scalability of EvoGraph is required to be explored to solve problems involving graphs with large number of nodes such as the Internet. Recent development in data compression of Internet graphs is heading toward this direction. For example, there is proposal on the formation of graph supernode by merging a number of nodes in the Internet graph [57] in order to reduce storage space and computational resource.

The elimination of isomorphic graph redundancy is another field to be explored to improve the efficiency of EvoGraph. This problem may cause unnecessary increase in size of search space such as node permutation redundancy in ANN [35]. The success on identifying isomorphic graphs depends on researches on isomorphic graph matching [56][58], which has been a challenging subject of research in the last three decades.. A simple algorithm on matching isomorphic graphs is yet to be devised. Recent researches reveal more and more *determined by spectrum* graphs, or DS-graphs [94]. This is encouraging because graph evolution can be greatly improved if redundant isomorphic graphs can be eliminated in the process of evolution by making use of the knowledge of DS-graphs.

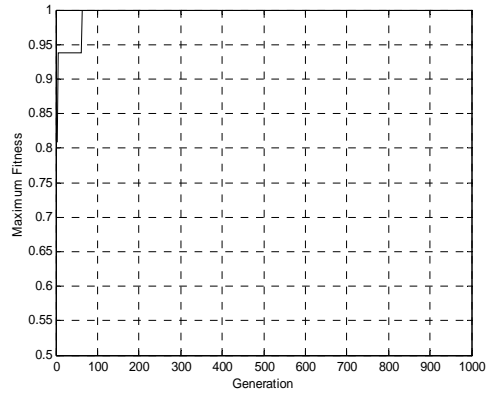
On the application side, EvoGraph is designed to solve problems involving graph topologies. EvoGraph is applied to architectural design, art creation, molecular design and communication network in this thesis. In a wider perspective, EvoGraph can be applied to problems that can be presented in graph such as critical paths, logistic networks, etc..

## Fitness Graphs on Best Performed Experiments on EvoGraph Operators and Conventional Evolution Operators

BEST PERFORMED EXPERIMENTS USING CONVENTIONAL EVOLUTION OPERATORS	BEST PERFORMED EXPERIMENTS USING EVOGRAPH OPERATORS
 <p data-bbox="235 989 773 1079">Target lattice graph evolution using conventional edge/node mutations (Max. fitness, generation) = ( 0.9482, 851)</p>	 <p data-bbox="797 989 1365 1108">Target lattice graph using EvoGraph <i>random crossover, number-of node, number-of-edge</i> mutations (Max. fitness, generation) = ( 1, 184)</p>
 <p data-bbox="235 1541 773 1631">Target ANN bipartite graph using conventional edge/node mutations (Max. fitness, generation) = ( 1, 166)</p>	 <p data-bbox="797 1541 1365 1661">Target ANN bipartite graph using EvoGraph <i>random crossover, number-of node, number-of-edge</i> mutations (Max. fitness, generation) = ( 1, 115)</p>



Target starlike tree evolution using standard GP crossover  
 (Max. fitness, generation) = (0.9602, 163)



Target starlike tree using EvoGraph *random crossover*  
 (Max. fitness, generation) = (1, 63)

## Experimental Results on Architectural Space Topology Evolution

### ADJACENCY MATRICES REPRESENTING OPTIMAL ARCHITECTURAL SPACE TOPOLOGIES GENERATED BY EXPERIMENTS

	10	1	3	5	4	6	7	2	9	8
10	-	3	3	2	3	1	3	3	0	3
1		-	1	0	0	0	0	2	3	0
3			-	0	0	3	0	0	3	0
5				-	1	0	0	0	3	1
4					-	2	0	0	0	3
6						-	0	0	3	0
7							-	0	3	0
2								-	3	3
9									-	0
8										-

Experiment 1 (Budget 30 to 34)

	9a	2	1	6	10	5	3	7	8	9b	4
9a	-	3	3	3	0	0	3	3	0	3	0
2		-	0	0	0	0	2	0	3	3	0
1			-	0	3	0	0	0	0	3	0
6				-	1	2	3	0	0	0	0
10					-	2	0	3	3	0	3
5						-	0	0	1	3	0
3							-	0	1	0	0
7								-	0	3	1
8									-	0	3
9b										-	3
4											-

Experiment 2 (Budget 35 to 39)

	8	2a	2b	7	5	4	6	3	1	9a	10	9b
8	-	3	3	0	1	3	0	1	0	0	3	0
2a		-	0	0	0	0	0	0	2	0	3	3
2b			-	0	0	0	0	2	0	0	3	3
7				-	0	1	0	0	0	3	3	3
5					-	0	2	0	0	0	2	3
4						-	2	0	0	3	0	0
6							-	0	0	3	1	0
3								-	0	3	3	0
1									-	3	3	3
9a										-	0	3
10											-	0
9b												-

Experiment 3 (Budget 40 to 44)

	9	3a	4a	7	2a	3b	2b	5	6	4b	8	1	10
9	-	3	3	0	3	3	0	3	3	0	0	0	0
3a		-	0	0	0	0	0	0	0	0	1	1	3
4a			-	1	0	0	0	0	0	0	3	0	3
7				-	0	0	0	0	0	0	0	0	3
2a					-	0	0	0	0	0	3	2	3
3b						-	2	0	3	0	1	0	0
2b							-	0	0	0	3	2	3
5								-	0	1	1	0	2
6									-	2	0	0	1
4b										-	3	0	3
8											-	0	3
1												-	3
10													-

Experiment 4 (Budget 45 to 49)

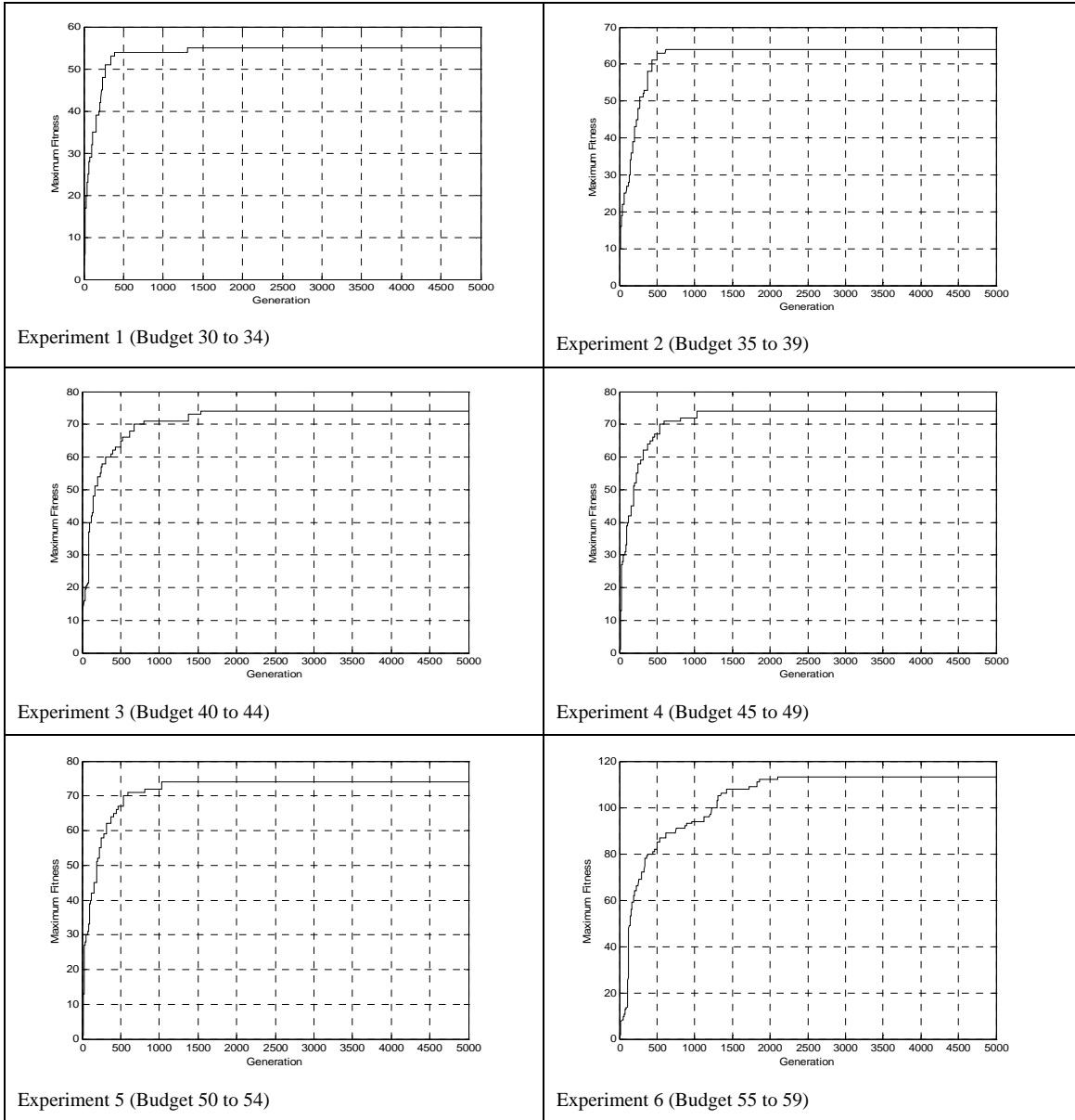
	9a	9b	6	3	9c	4	2	9d	7	9e	5	9f	8	1	10
9a	-	3	3	3	3	0	0	0	3	0	0	0	0	3	0
9b		-	3	3	0	3	3	0	0	0	0	3	0	0	0
6			-	3	0	0	0	3	0	0	0	0	0	0	0
3				-	0	0	0	0	0	3	0	0	0	0	0
9c					-	3	0	3	3	0	3	3	0	0	0
4						-	0	0	0	3	0	0	0	0	3
2							-	3	0	3	0	0	3	0	0
9d								-	3	0	3	0	0	3	0
7									-	3	0	3	0	0	3
9e										-	3	3	0	0	0
5											-	3	0	0	0
9f												-	0	3	0
8													-	0	3
1														-	3
10															-

Experiment 5 (Budget 50 to 54)

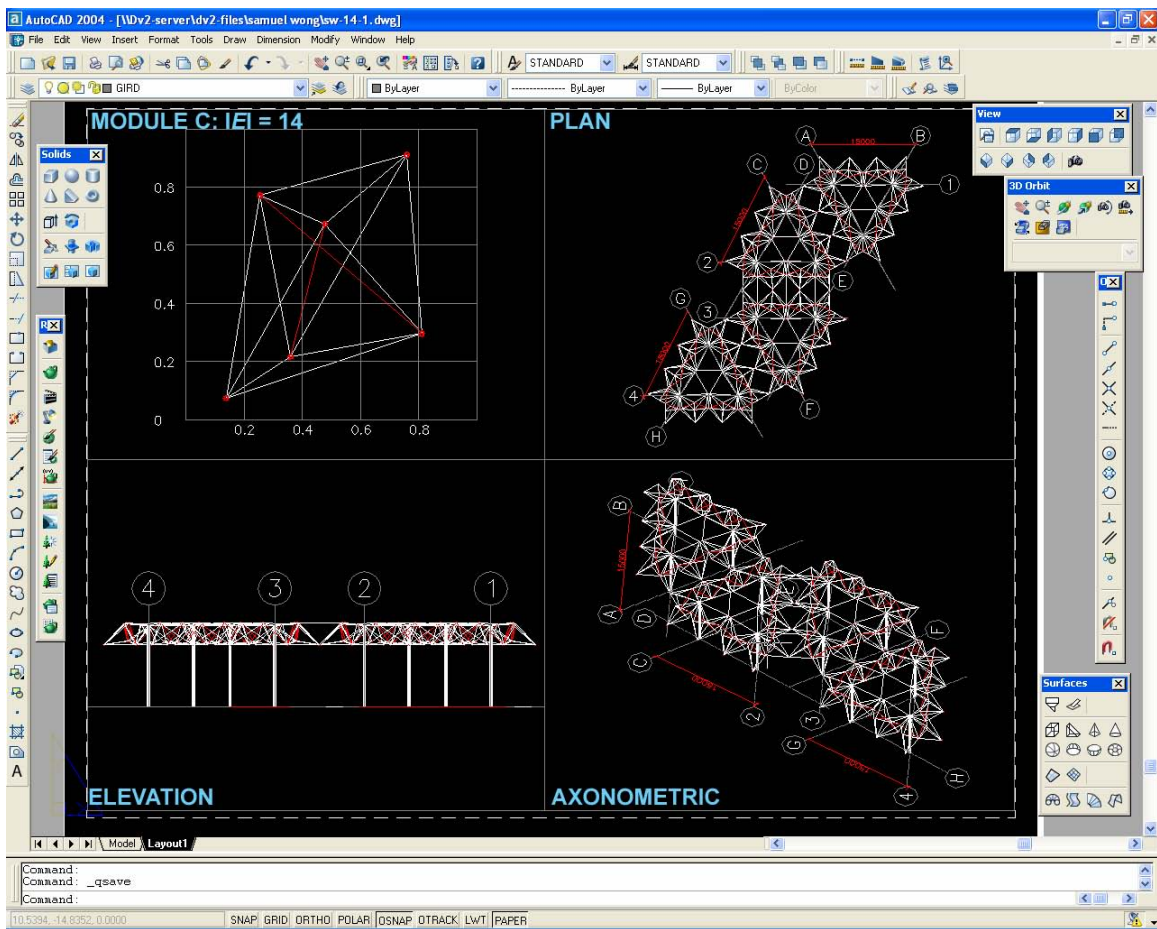
	10	2	9a	3a	3b	6	4a	9b	1	9c	9d	8	4b	5	9e	7
10	-	3	0	3	3	1	3	0	3	0	0	3	3	2	0	3
2		-	0	0	2	0	0	0	0	0	3	0	0	0	3	0
9a			-	0	0	3	0	0	3	3	0	0	0	3	3	3
3a				-	0	3	0	0	0	0	3	1	0	0	0	0
3b					-	0	0	0	0	3	0	0	0	0	3	0
6						-	0	0	0	0	3	0	0	0	0	0
4a							-	0	0	0	3	3	0	0	0	1
9b								-	3	3	3	0	3	3	0	3
1									-	0	0	0	0	0	3	0
9c										-	0	0	0	3	0	3
9d											-	0	0	0	0	3
8												-	0	0	0	0
4b													-	0	3	1
5														-	0	0
9e															-	3
7																-

Experiment 6 (Budget 55 to 59)

# FITNESS GRAPHS ON ARCHITECTURAL TOPOLOGY EVOLUTION

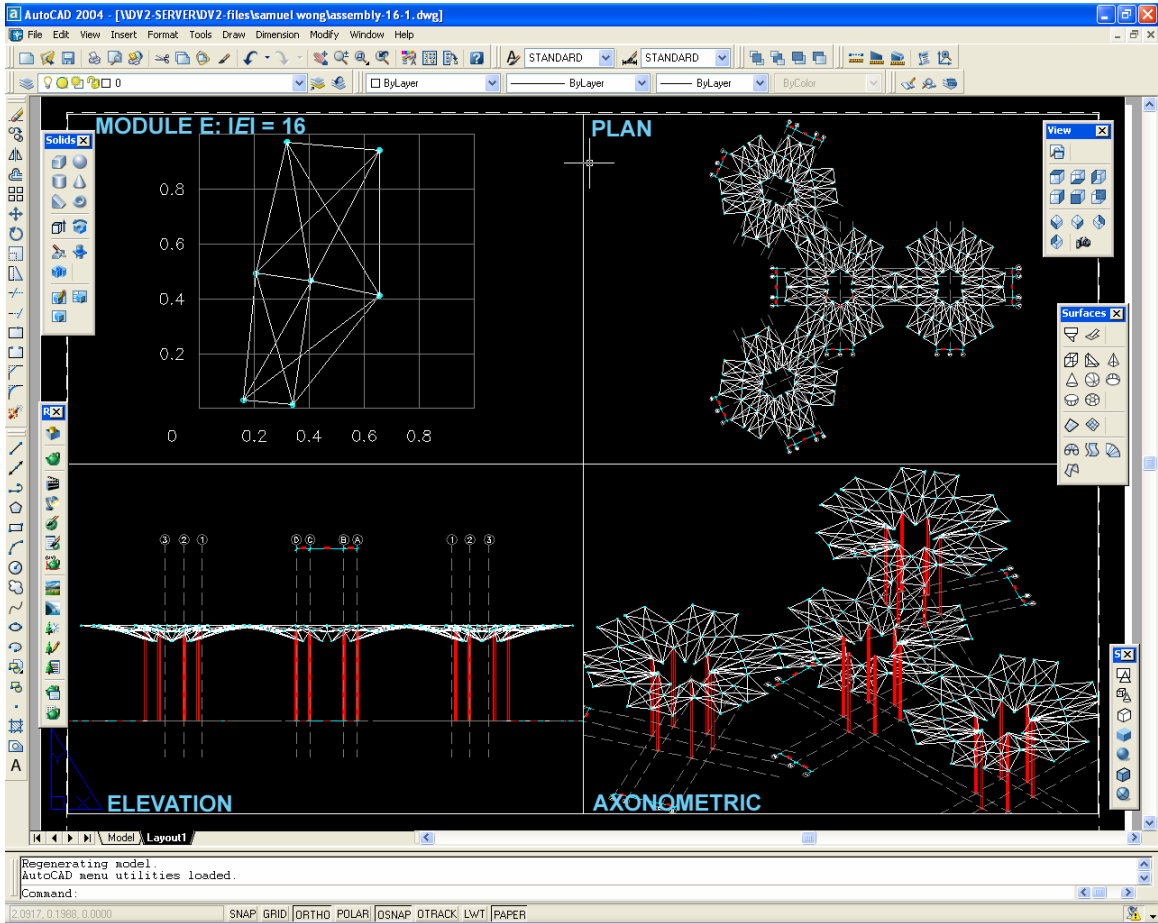


## Repetitive Space Frame Modules

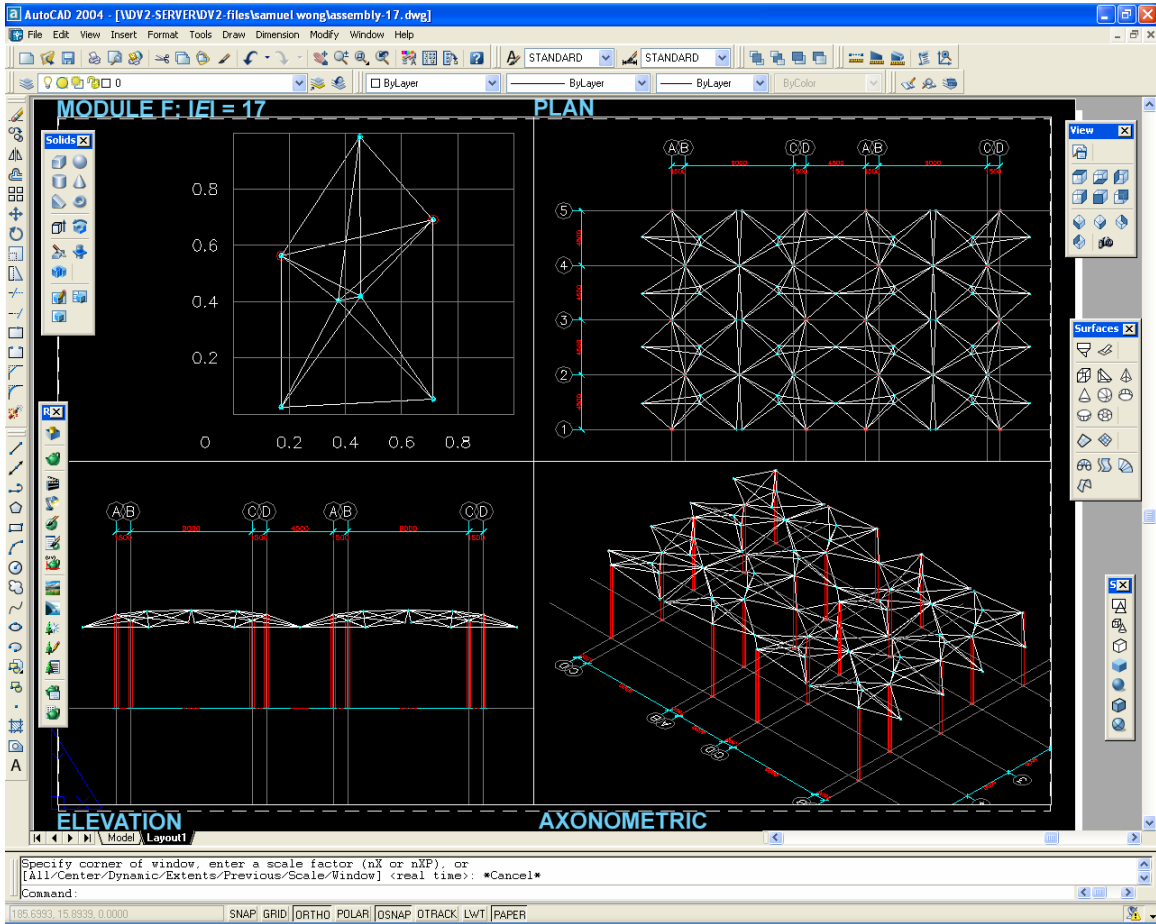


Module C with 14 edges

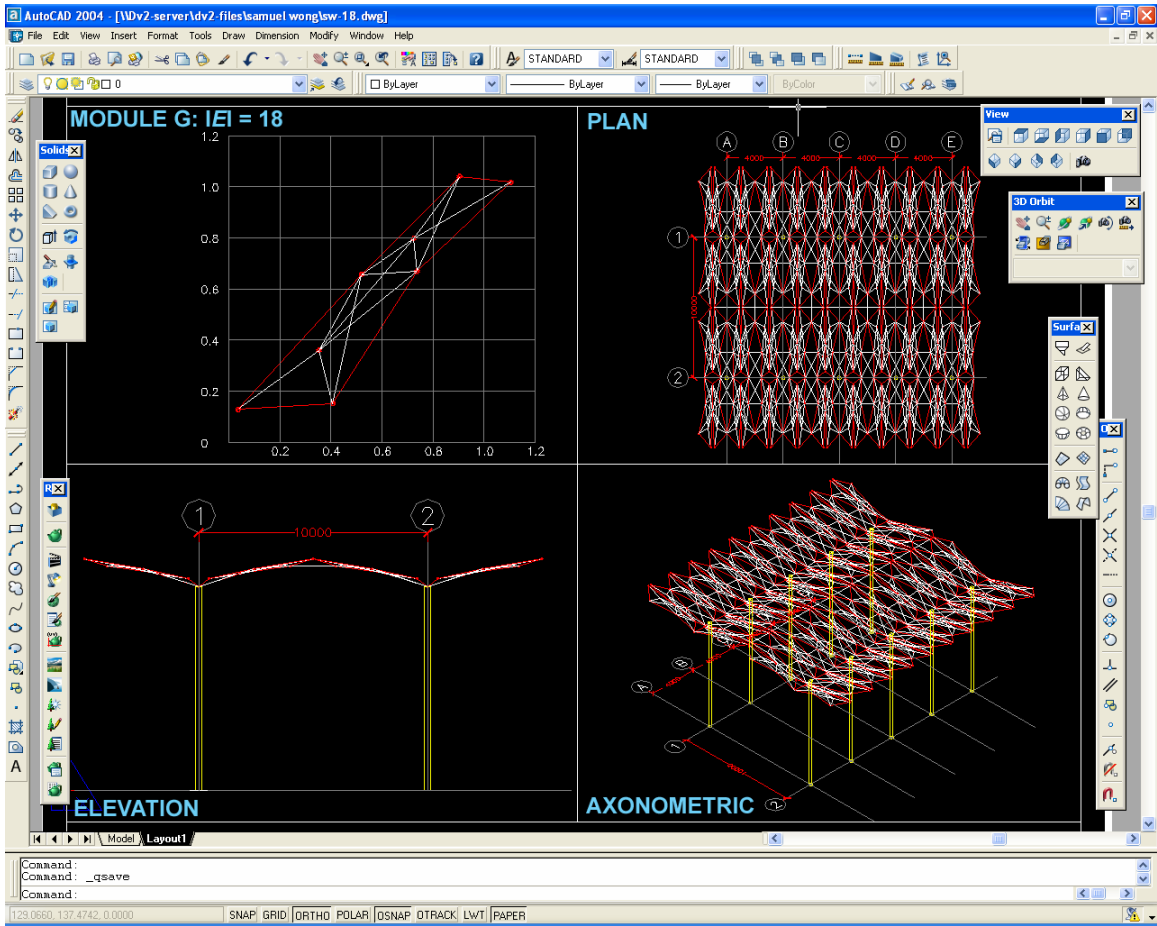




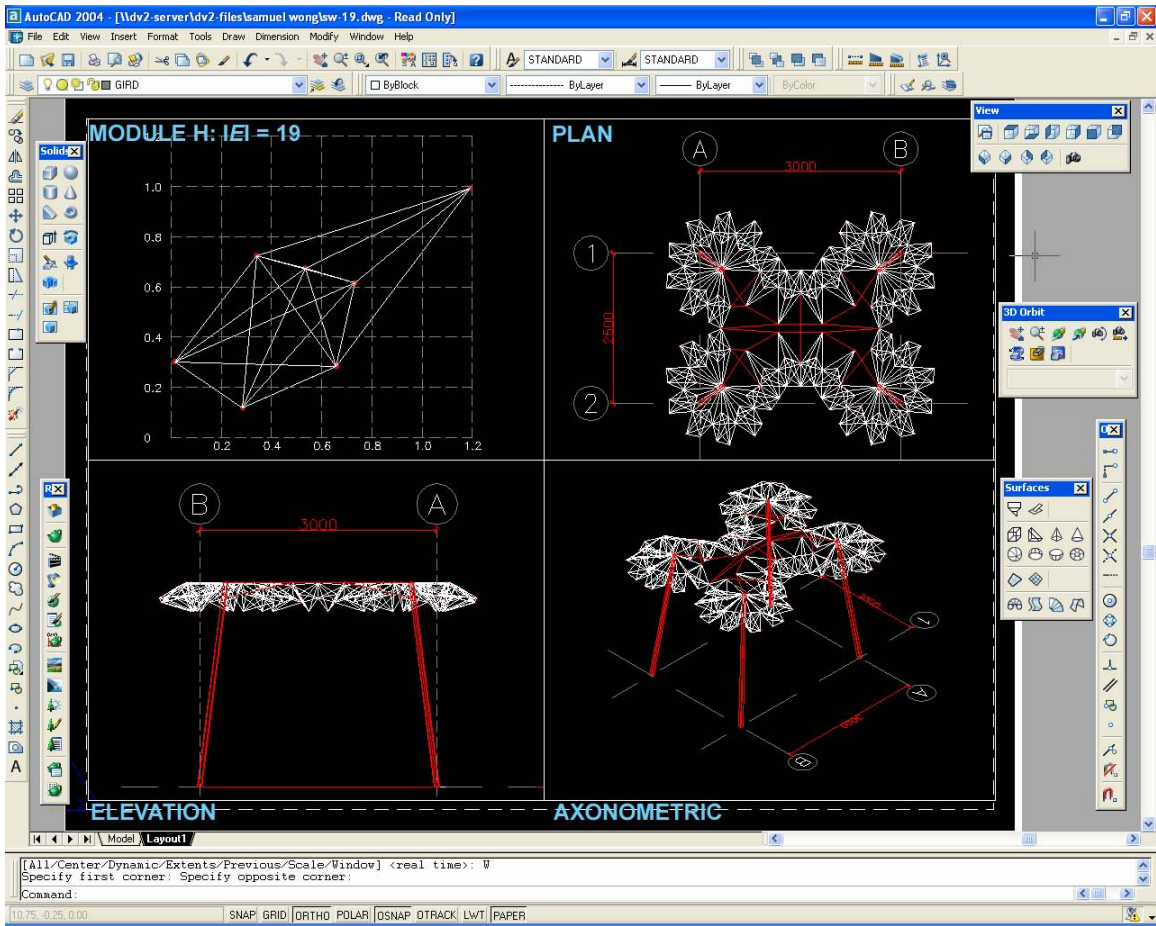
**Module D with 15 edges**



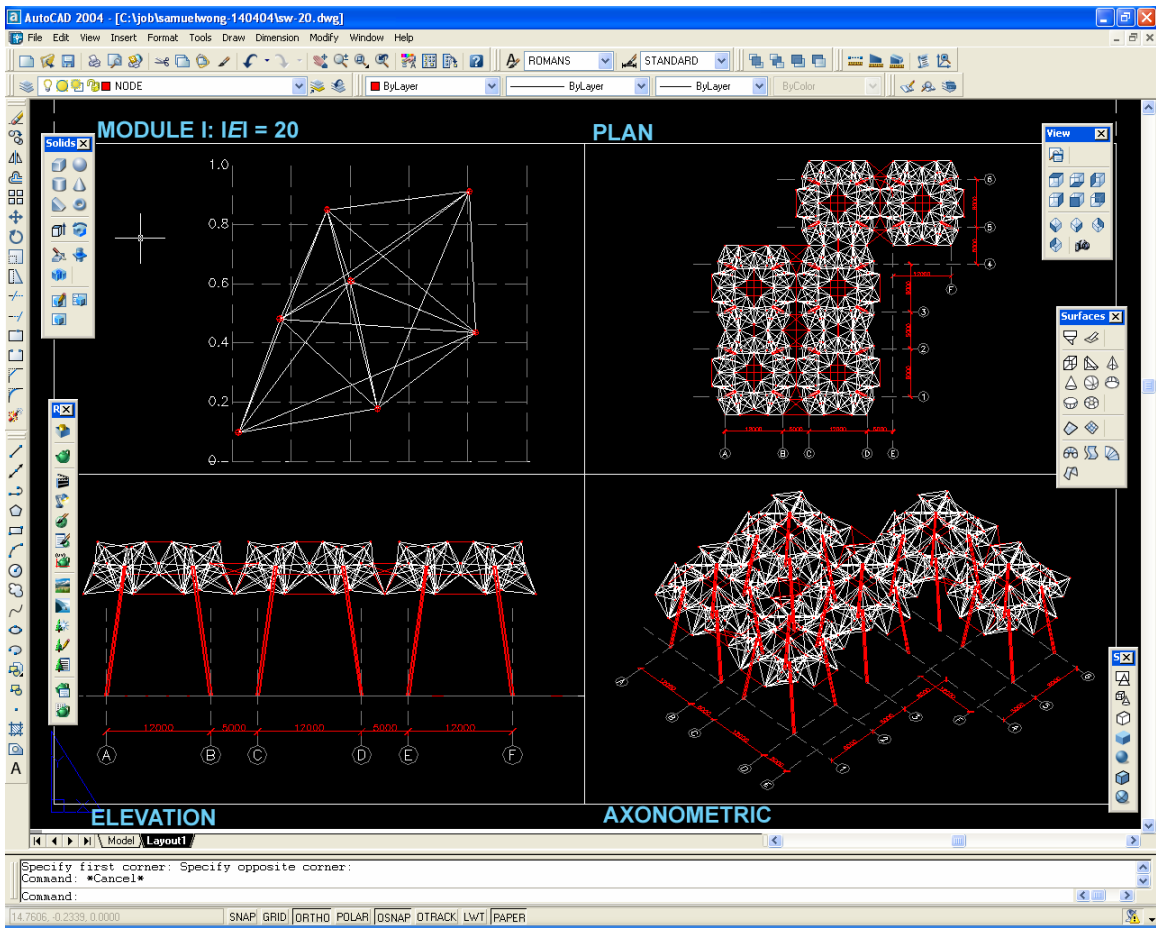
Module F with 17 edges



**Module G with 18 edges**



**Module H with 19 edges**

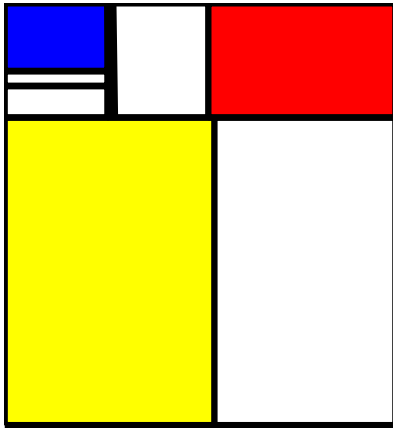


**Module I with 20 edges**

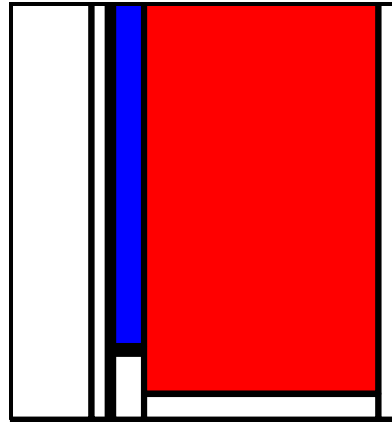
---

## Mondrian Paintings Evolved by Tree Evolution

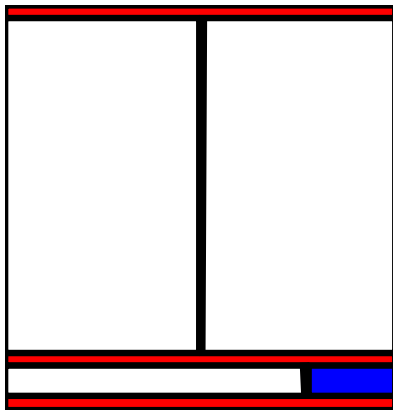
---



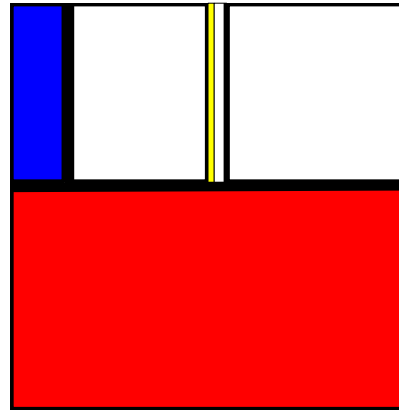
Experiment 1 with Mix 5



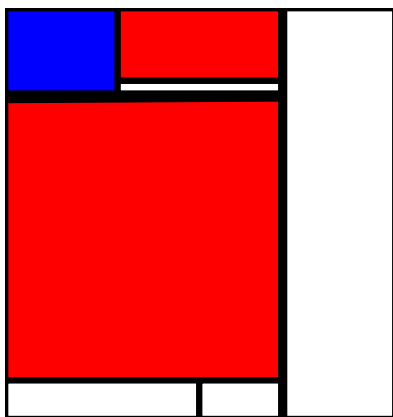
Experiment 2 with Mix 6



Experiment 3 with Mix 8



Experiment 4 with Mix 5

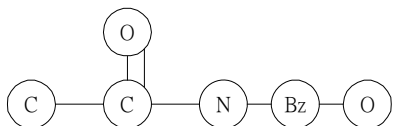


Experiment 5 with Mix 5

---

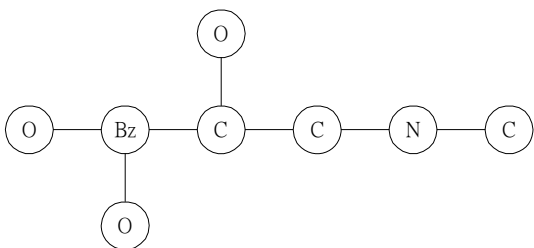
## Target Hydrogen Depleted Molecules and Corresponding Molecular Graphs

---



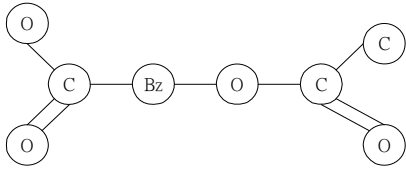
	12	12	16	16	14	72
12	0	1	0	0	0	0
12	1	0	0	2	1	0
16	0	0	0	0	0	0
16	0	2	0	0	0	1
14	0	1	0	0	0	1
72	0	0	0	1	1	0

### 1. tylenol



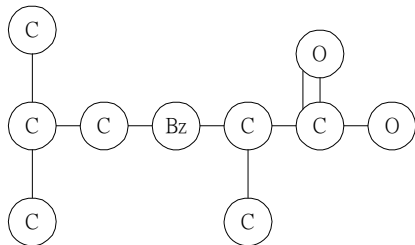
	12	12	12	16	16	16	14	72
12	0	1	0	0	0	1	0	1
12	1	0	0	0	0	0	1	0
12	0	0	0	0	0	0	1	0
16	0	0	0	0	0	0	0	1
16	0	0	0	0	0	0	0	1
16	1	0	0	0	0	0	0	0
14	0	1	1	0	0	0	0	0
72	1	0	0	1	1	0	0	0

### 2. adrenaline



**3. aspirin**

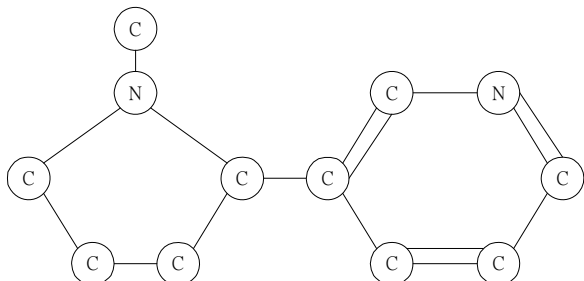
	12	12	12	16	16	16	16	72
12	0	0	0	2	1	0	0	1
12	0	0	1	0	0	1	2	0
12	0	1	0	0	0	0	0	0
16	2	0	0	0	0	0	0	0
16	1	0	0	0	0	0	0	0
16	0	1	0	0	0	0	0	1
16	0	2	0	0	0	0	0	0
72	1	0	0	0	0	1	0	0



**4. ibuprofen**

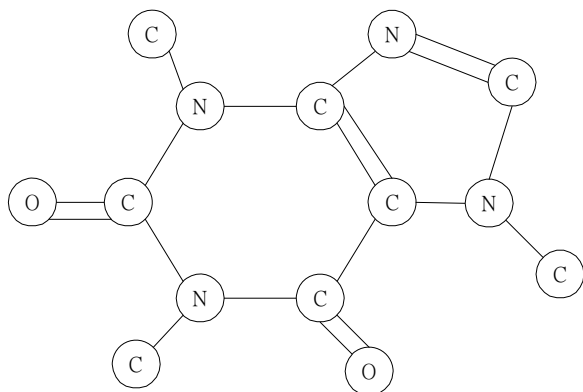
	12	12	12	12	12	12	12	16	16	72
12	0	1	0	0	0	0	0	0	0	0
12	1	0	1	1	0	0	0	0	0	0
12	0	1	0	0	0	0	0	0	0	0
12	0	1	0	0	0	0	0	0	0	1
12	0	0	0	0	0	1	1	0	0	1
12	0	0	0	0	1	0	0	0	0	0
12	0	0	0	0	1	0	0	2	1	0
16	0	0	0	0	0	0	2	0	0	0
16	0	0	0	0	0	0	1	0	0	0
72	0	0	0	1	1	0	0	0	0	0





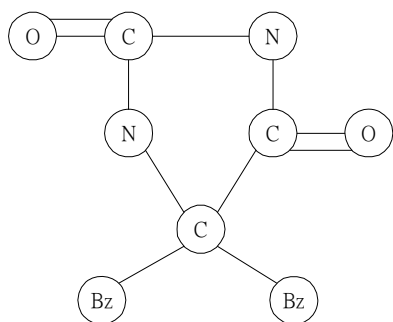
	12	12	12	12	12	12	12	12	12	12	14	14
12	0	0	0	0	0	0	0	0	0	0	1	0
12	0	0	1	0	0	0	0	0	0	0	1	0
12	0	1	0	1	0	0	0	0	0	0	0	0
12	0	0	1	0	1	0	0	0	0	0	1	0
12	0	0	0	1	0	1	0	0	0	0	0	0
12	0	0	0	0	1	0	1	0	0	2	0	0
12	0	0	0	0	0	1	0	2	0	0	0	0
12	0	0	0	0	0	0	2	0	1	0	0	0
12	0	0	0	0	0	0	0	1	0	0	0	2
12	0	0	0	0	0	2	0	0	0	0	0	1
14	1	1	0	1	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	2	1	0	0

### 5. nicotine



	12	12	12	12	12	12	12	12	16	16	14	14	14	14
12	0	0	0	0	0	0	0	0	0	0	1	0	0	0
12	0	0	0	0	0	0	0	0	2	0	1	1	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0
12	0	0	0	0	1	0	0	0	0	2	0	1	0	0
12	0	0	0	1	0	2	0	0	0	0	0	0	1	0
12	0	0	0	0	2	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	1	2
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0
16	0	2	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	2	0	0	0	0	0	0	0	0	0	0
14	1	1	0	0	0	1	0	0	0	0	0	0	0	0
14	0	1	1	1	0	0	0	1	0	0	0	0	0	0
14	0	0	0	0	1	0	1	1	0	0	0	0	0	0
14	0	0	0	0	0	1	2	0	0	0	0	0	0	0

## 6. caffeine



	12	12	12	16	16	14	14	72	72
12	0	0	0	2	0	1	1	0	0
12	0	0	1	0	0	1	0	1	1
12	0	1	0	0	2	0	1	0	0
16	2	0	0	0	0	0	0	0	0
16	0	0	2	0	0	0	0	0	0
14	1	1	0	0	0	0	0	0	0
14	1	0	1	0	0	0	0	0	0
72	0	1	0	0	0	0	0	0	0
72	0	1	0	0	0	0	0	0	0

## 7. dilantin

---

## Underlay Networks Cost Matrices with Approximate Minimum Total Cost Tree (AMTCT) and Cost Savings of each AMTCT

---

Note: the cost links of the AMTCT cost matrices found by EvoGraph are underlined and bold

$|N|=8$

0	1	2	3	4	5	6	7	8
1	0	88	<b><u>2</u></b>	<b><u>40</u></b>	<b><u>73</u></b>	0	65	81
2	88	0	0	34	0	0	99	84
3	9	0	0	47	81	<b><u>33</u></b>	34	<b><u>60</u></b>
4	40	<b><u>34</u></b>	47	0	70	31	0	0
5	73	0	81	70	0	56	94	50
6	0	0	33	31	56	0	<b><u>11</u></b>	0
7	65	99	34	0	94	11	0	91
8	81	84	60	0	50	0	91	0

$|N|=10$

0	1	2	3	4	5	6	7	8	9	10
1	0	91	86	<b><u>50</u></b>	0	0	48	<b><u>10</u></b>	97	0
2	91	0	0	17	92	79	35	47	0	0
3	86	0	0	92	<b><u>10</u></b>	57	39	0	31	0
4	50	<b><u>17</u></b>	92	0	0	0	0	60	79	33
5	0	92	10	0	0	<b><u>37</u></b>	68	0	70	0
6	0	79	57	0	37	0	0	0	0	71
7	48	35	39	0	68	0	0	34	0	45
8	10	47	0	60	0	0	<b><u>34</u></b>	0	<b><u>22</u></b>	<b><u>40</u></b>
9	97	0	<b><u>31</u></b>	79	70	0	0	22	0	84
10	0	0	0	33	0	71	45	40	84	0

$|N|=12$

0	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	<b><u>7</u></b>	0	<b><u>18</u></b>	<b><u>24</u></b>	47	83	0	0	0	77
2	0	0	45	0	8	0	32	91	<b><u>33</u></b>	0	0	100
3	7	45	0	0	0	75	80	<b><u>14</u></b>	0	0	0	79
4	0	0	0	0	84	56	42	69	41	98	0	0
5	18	<b><u>8</u></b>	0	84	0	33	0	0	0	0	77	76
6	24	0	75	<b><u>56</u></b>	33	0	<b><u>32</u></b>	0	0	0	75	0
7	47	32	80	42	0	32	0	0	71	0	0	0
8	83	91	14	69	0	0	0	0	0	<b><u>12</u></b>	0	96
9	0	33	0	41	0	0	71	0	0	62	61	0
10	0	0	0	98	0	0	0	12	62	0	<b><u>65</u></b>	<b><u>45</u></b>
11	0	0	0	0	77	75	0	0	61	65	0	0
12	77	100	79	0	76	0	0	96	0	45	0	0

$|N|=14$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	<u>3</u>	69	41	<u>38</u>	0	0	67	0	<u>3</u>	0	0	0
2	0	0	8	94	0	90	0	0	0	79	0	0	72	33
3	3	<u>8</u>	0	<u>64</u>	0	0	53	0	88	0	0	0	78	0
4	69	94	64	0	80	0	0	0	0	0	0	93	0	94
5	41	0	0	80	0	63	2	0	0	51	0	0	19	0
6	38	90	0	0	63	0	0	0	0	<u>25</u>	95	0	79	0
7	0	0	53	0	<u>2</u>	0	0	0	<u>32</u>	0	31	0	0	0
8	0	0	0	0	0	0	0	0	49	0	44	84	7	<u>11</u>
9	67	0	88	0	0	0	32	49	0	57	0	85	0	0
10	0	79	0	0	51	25	0	0	57	0	0	0	0	0
11	3	0	0	0	0	95	<u>31</u>	44	0	0	0	48	<u>32</u>	0
12	0	0	0	93	0	0	0	84	85	0	48	0	0	16
13	0	72	78	0	19	79	0	<u>7</u>	0	0	32	0	0	0
14	0	33	0	94	0	0	0	11	0	0	0	<u>16</u>	0	0

$|N|=16$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	96	<u>7</u>	91	0	71	0	0	0	0	<u>15</u>	0	0	0	72
2	0	0	0	79	34	<u>25</u>	0	0	39	0	55	0	0	18	0	0
3	96	0	0	0	94	0	0	59	0	0	0	0	16	0	50	<u>44</u>
4	7	79	0	0	0	0	0	0	<u>49</u>	0	92	0	<u>14</u>	0	0	72
5	91	34	94	0	0	0	0	0	0	0	56	30	18	0	0	0
6	0	25	0	0	0	0	55	89	46	83	0	0	0	0	0	0
7	71	0	0	0	0	55	0	0	50	0	0	13	0	49	<u>31</u>	0
8	0	0	59	0	0	89	0	0	0	55	0	0	0	53	81	86
9	0	39	0	49	0	46	50	0	0	<u>34</u>	0	0	63	0	0	0
10	0	0	0	0	0	83	0	55	34	0	0	0	83	0	59	0
11	0	55	0	92	56	0	0	0	0	0	0	28	0	84	0	87
12	15	0	0	0	30	0	<u>13</u>	0	0	0	<u>28</u>	0	83	<u>22</u>	0	0
13	0	0	<u>16</u>	14	<u>18</u>	0	0	0	63	83	0	83	0	0	0	0
14	0	<u>18</u>	0	0	0	0	49	<u>53</u>	0	0	84	22	0	0	0	0
15	0	0	50	0	0	0	31	81	0	59	0	0	0	0	0	93
16	72	0	44	72	0	0	0	86	0	0	87	0	0	0	93	0

$|N|=18$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	8
1	0	0	<u>3</u>	0	0	0	90	0	0	0	0	0	97	77	84	0	<u>11</u>	0
2	0	0	0	1	0	78	0	59	34	0	0	86	0	0	0	0	0	80
3	3	0	0	0	62	0	77	0	0	<u>10</u>	0	94	0	74	0	0	0	0
4	0	<u>1</u>	0	0	49	0	0	2	0	0	0	0	11	0	0	0	0	0
5	0	0	62	49	0	0	<u>54</u>	0	0	19	0	0	0	0	38	0	22	0
6	0	78	0	0	0	0	0	0	87	0	34	0	<u>19</u>	0	0	0	19	52
7	90	0	77	0	54	0	0	0	0	0	0	34	98	0	64	0	0	0
8	0	59	0	<u>2</u>	0	0	0	0	0	0	0	0	52	<u>6</u>	0	0	90	13
9	0	34	0	0	0	87	0	0	0	10	85	74	0	0	0	0	0	0
10	0	0	10	0	<u>19</u>	0	0	0	<u>10</u>	0	<u>5</u>	0	0	0	90	0	92	0
11	0	0	0	0	0	34	0	0	85	5	0	0	0	0	0	<u>15</u>	0	0
12	0	86	94	0	0	0	34	0	74	0	0	0	0	82	0	57	0	0
13	97	0	0	11	0	<u>19</u>	98	52	0	0	0	0	0	0	0	97	0	0
14	77	0	74	0	0	0	0	6	0	0	0	82	0	0	0	75	0	96
15	84	0	0	0	38	0	64	0	0	90	0	0	0	0	0	0	45	80
16	0	0	0	0	0	0	0	0	0	0	15	<u>57</u>	97	75	0	0	0	<u>10</u>
17	11	0	0	0	22	19	0	90	0	92	0	0	0	0	<u>45</u>	0	0	0
18	0	80	0	0	0	52	0	<u>13</u>	0	0	0	0	0	96	80	10	0	0

$|N|=20$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	0	0	75	0	0	0	0	0	88	<u>48</u>	0	0	0	0	<u>40</u>	0	96	0	0
2	0	0	0	0	0	<u>14</u>	0	19	0	0	57	71	0	93	0	86	0	34	0	0
3	0	0	0	0	28	0	0	<u>5</u>	0	56	80	0	14	63	0	0	0	56	0	88
4	75	0	0	0	0	0	0	64	<u>7</u>	0	0	27	9	0	57	0	0	41	44	0
5	0	0	28	0	0	0	39	100	0	44	0	45	0	0	<u>36</u>	0	30	0	0	39
6	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	75	0	0	0	0
7	0	0	0	0	39	0	0	0	0	37	31	0	0	0	0	88	0	0	0	0
8	0	<u>19</u>	5	64	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	7	0	0	0	0	0	0	0	28	0	88	0	0	0	66	0	0
10	88	0	56	0	44	0	37	0	0	0	11	15	0	0	0	0	61	65	0	0
11	48	57	80	0	0	0	<u>31</u>	0	0	<u>11</u>	0	0	<u>23</u>	0	0	73	0	0	<u>6</u>	0
12	0	71	0	27	45	0	0	0	28	15	0	0	0	0	0	12	0	0	0	0
13	0	0	<u>14</u>	<u>9</u>	0	0	0	0	0	0	23	0	0	0	0	0	0	0	0	0
14	0	93	63	0	0	0	0	0	88	0	0	0	0	0	0	0	43	0	0	0
15	0	0	0	57	36	0	0	0	0	0	0	0	0	0	0	90	0	0	0	0
16	40	86	0	0	0	75	88	0	0	0	73	<u>12</u>	0	0	90	0	85	0	0	0
17	0	0	0	0	<u>30</u>	0	0	0	0	61	0	0	0	<u>43</u>	0	85	0	<u>42</u>	2	<u>2</u>
18	96	34	56	41	0	0	0	0	66	65	0	0	0	0	0	0	42	0	0	0
19	0	0	0	44	0	0	0	0	0	0	6	0	0	0	0	0	<u>2</u>	0	0	0
20	0	0	88	0	39	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0

|N|=22

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	0	0	86	50	0	0	98	<u>34</u>	0	<u>16</u>	0	0	0	0	0	52	0	0	0	0	<u>19</u>	<u>76</u>
2	0	0	29	76	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	86	<u>29</u>	0	0	13	0	0	48	0	0	31	0	0	0	0	0	0	0	37	<u>8</u>	0	93
4	50	76	0	0	0	35	0	0	<u>14</u>	17	0	0	<u>10</u>	0	0	0	94	0	49	0	0	0
5	0	0	<u>13</u>	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	35	0	0	0	0	0	85	<u>27</u>	0	0	0	0	6	0	0	0	0	0	0
7	98	0	0	0	<u>6</u>	0	0	0	0	0	0	0	0	0	<u>32</u>	1	<u>68</u>	0	0	42	41	92
8	34	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	14	0	0	0	0	0	0	0	0	66	0	0	58	0	0	0	0	0	0
10	16	0	0	<u>17</u>	0	85	0	0	0	0	0	53	0	0	0	<u>22</u>	0	<u>68</u>	61	73	0	0
11	0	0	31	0	0	27	0	0	0	0	0	0	0	35	0	0	78	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	53	0	0	6	0	0	59	0	0	0	0	0	0
13	0	0	0	10	0	0	0	0	66	0	0	<u>6</u>	0	85	56	0	0	58	<u>38</u>	66	0	0
14	0	0	0	0	0	0	0	0	0	0	35	0	85	0	0	38	0	0	0	0	0	0
15	0	0	0	0	0	0	32	0	0	0	0	0	56	0	0	0	0	0	0	0	0	0
16	52	0	0	0	0	<u>6</u>	<u>1</u>	0	58	22	0	59	0	<u>38</u>	0	0	0	0	0	0	0	0
17	0	0	0	94	0	0	68	0	0	0	78	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	68	0	0	58	0	0	0	0	0	0	0	0	0
19	0	0	37	49	0	0	0	0	0	61	0	0	38	0	0	0	0	0	0	0	0	0
20	0	0	8	0	0	0	42	0	0	73	0	0	66	0	0	0	0	0	0	0	0	0
21	19	0	0	0	0	0	41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	76	0	93	0	0	0	92	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$|N|=24$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	
1	0	0	0	0	38	0	0	95	0	0	<u>40</u>	0	0	<u>2</u>	<u>14</u>	0	0	0	49	<u>15</u>	0	90	0	0	
2	0	0	0	0	0	98	0	96	0	0	0	0	0	0	35	39	0	0	0	0	0	0	0	0	
3	0	0	0	92	0	0	0	0	0	0	0	0	0	41	70	0	0	<u>20</u>	0	0	22	64	28	<u>24</u>	
4	0	0	92	0	18	0	0	<u>36</u>	0	0	77	59	0	0	75	0	0	0	0	0	0	0	0	38	0
5	38	0	0	<u>18</u>	0	0	0	0	54	<u>41</u>	0	0	0	33	0	0	0	65	0	0	0	42	0	88	
6	0	98	0	0	0	0	0	0	30	91	0	0	0	0	0	0	0	0	14	58	26	3	0	65	
7	0	0	0	0	0	0	0	93	0	0	0	0	0	0	0	0	0	0	0	27	0	0	0	0	
8	95	96	0	36	0	0	93	0	0	36	0	99	60	0	0	0	59	0	0	0	0	0	0	0	
9	0	0	0	0	54	30	0	0	0	0	26	0	0	0	0	0	0	0	0	0	0	0	25	0	
10	0	0	0	0	41	91	0	36	0	0	74	0	0	0	48	0	0	0	0	0	0	0	0	0	
11	40	0	0	77	0	0	0	0	26	74	0	67	46	0	0	0	0	0	0	0	<u>10</u>	31	0	0	
12	0	0	0	59	0	0	0	99	0	0	67	0	0	68	0	0	81	0	99	0	50	66	0	0	
13	0	0	0	0	0	0	0	60	0	0	46	0	0	0	15	0	0	0	0	0	0	0	21	0	
14	2	0	<u>41</u>	0	<u>33</u>	0	0	0	0	0	0	<u>68</u>	0	0	0	0	0	0	0	0	0	0	0	0	
15	14	<u>35</u>	70	75	0	0	0	0	0	0	0	0	<u>15</u>	0	0	0	0	0	<u>15</u>	0	0	56	<u>15</u>	0	
16	0	39	0	0	0	0	0	0	0	48	0	0	0	0	0	<u>42</u>	39	0	39	0	0	0	0	0	
17	0	0	0	0	0	0	0	59	0	0	0	81	0	0	0	42	0	0	0	0	0	0	88	0	
18	0	0	20	0	65	0	0	0	0	0	0	0	0	0	<u>39</u>	0	0	0	0	0	0	0	0	0	
19	49	0	0	0	0	14	0	0	0	0	0	99	0	0	15	0	0	0	0	56	0	0	13	0	
20	15	0	0	0	0	58	<u>27</u>	0	0	0	0	0	0	0	39	0	0	56	0	41	<u>20</u>	78	0	0	
21	0	0	22	0	0	26	0	0	0	0	10	50	0	0	0	0	0	0	0	41	0	0	0	0	
22	90	0	64	0	42	<u>3</u>	0	0	0	0	31	66	0	0	56	0	0	0	0	20	0	0	0	0	
23	0	0	28	38	0	0	0	0	<u>25</u>	0	0	0	21	0	15	0	88	0	13	78	0	0	0	0	
24	0	0	24	0	88	65	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	



COST SAVING RESULTS OF AMTCTS ON EACH UNDERLAY NETWORK

**|V|=8**

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
SPT	350	283	633
AMTCT	360	260	620
net	-10	23	13
% saving			2%

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
MST	530	224	754
AMTCT	360	260	620
net	170	-36	134
% saving			18%

**|V|=10**

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
SPT	489	281	770
AMTCT	499	251	750
net	-10	30	20
% saving			3%

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
MST	636	229	865
AMTCT	499	251	750
net	137	-22	115
% saving			13%

**|V|=12**

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
SPT	487	373	860
AMTCT	500	314	814
net	-13	59	46
% saving			5%

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
MST	544	295	839
AMTCT	500	314	814
net	44	-19	25
% saving			3%

**|V|=14**

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
SPT	493	326	819
AMTCT	520	272	792
net	-27	54	27
% saving			3%

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
MST	600	259	859
AMTCT	520	272	792
net	80	-13	67
% saving			8%

**|V|=16**

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
SPT	729	415	1144
AMTCT	738	387	1125
net	-9	28	19
% saving			2%

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
MST	814	377	1191
AMTCT	738	387	1125
net	76	-10	66
% saving			6%

**|V|=18**

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
SPT	714	355	1069
AMTCT	722	299	1021
net	-8	56	48
% saving			4%

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
MST	844	261	1105
AMTCT	722	299	1021
net	122	-38	84
% saving			8%

**|V|=20**

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
SPT	1470	634	2104
AMTCT	1496	394	1890
net	-26	240	214
% saving			10%

**|V|=22**

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
SPT	1180	571	1751
AMTCT	1184	548	1732
net	-4	23	19
% saving			1%

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
MST	2083	351	2434
AMTCT	1496	432	1928
net	587	-81	506
% saving			21%

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
MST	1245	534	1779
AMTCT	1184	548	1732
net	61	-14	47
% saving			3%

**|V|=24**

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
SPT	1072	622	1694
AMTCT	1072	598	1670
net	0	24	24
% saving			1%

	<u>TPC</u>	<u>TEC</u>	<u>TC</u>
MST	1482	534	2016
AMTCT	1072	598	1670
net	410	-64	346
% saving			17%

---

## References

---

- [1] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston, MA: Addison-Wesley, 1989.
- [2] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA: The MIT Press, 1994.
- [3] J. R. Koza, "Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems," *Stanford University Computer Science Department technical report STAN-CS-90-1314*, June 1990.
- [4] X. Yao, "Evolving artificial neural networks," in *Proceedings of the IEEE*, vol. 87, no.9, pp.1423-1447, September 1999.
- [5] J. L. Gross, *Graph Theory And Its Applications*. Boca-Raton:FL, CRC Press, 1999.
- [6] L.R. Foulds, *Graph Theory Applications*, New York: NY, Springer-Verlag, 1992.
- [7] S. Yin and J. Cagan, "An extended pattern search algorithm for three-dimensional component layout," *Transactions of the ASME*, vol.122, pp. 102–108, 2000.
- [8] T. Ito, "A genetic algorithm approach to piping route path planning," *Journal of Intelligent Manufacturing*, vol. 10, pp. 103-104, 1999.
- [9] T. Koide and S. Wakabayashi, "A timing-driven floor planning algorithm with the Elmore delay model for building block layout," *Integration: the VLSI Journal*, vol.27, pp. 57–76, 1999.
- [10] P. H. Levin, "Use of graphs to decide the optimum layout of buildings," *Architect*, vol. 14, pp. 809–815, Oct. 1964.
- [11] B. Medjdoub and B. Yannou, "Separating topology and geometry in space planning," *Computer-Aided Design*, vol. 32, pp. 39–61, 2000.
- [12] F. Gruau, "Cellular encoding as a graph grammar," *IEE Colloquium on Grammatical Inference: Theory, Applications and Alternatives*, pp. 17/1 – 17/10, 22-23 Apr 1993
- [13] S. Luke, "Two fast tree-creation algorithms for genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 4 , Issue: 3, pp. 274 – 283, Sept. 2000
- [14] J.R. Koza and J.P. Rice, "Genetic generation of both the weights and architecture for a neural network," *IJCNN-91-Seattle International Joint Conference on Neural*

*Networks*, vol. 2, pp397 – 404 , 8-14 July 1991

- [15] S. Luke and L. Spector, "Evolving Graphs and Networks with Edge Encoding: Preliminary Report," in *Late-breaking Papers of the Genetic Programming (GP96) Conference*, Stanford, 1996.
- [16] N.Y. Nikolaev and H. Iba, "Learning polynomial feedforward neural networks by genetic programming and backpropagation," *IEEE Transactions on Neural Networks*, vol. 14 , Issue: 2, pp. 337 – 350, March 2003.
- [17] B. Ombuki, M. Nakamura, Z. Nakao, and K. Onaga, "Evolutionary Computation for Topological Optimization of 3-Connected Computer Networks," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 1999*, pp. 659-664, Oct 12-15, 1999.
- [18] A.Globus, J. Lawton, and T. Wipke, "Automatic molecular design using evolutionary techniques", *Nanotechnology*, vol.10,pp.290-299, 1999.
- [19] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, 8(3):694-713, May 1997
- [20] M. W. Hwang, J. Y. Choi, and J. Park, "Evolutionary projection neural networks," in *Proceedings of IEEE International Conference in Evolutionary Computation 1997 ICEC'97*, pp. 667–671.
- [21] S. Nolfi, D. Parisi, "Genotypes for neural networks" in M. A. Arbib, *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA: MIT Press/Bradford Books, 1995.
- [22] F. Gruau, "Genetic synthesis of boolean neural networks with a cell rewriting developmental process," in *Proceedings of International Workshop Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, D. Whitley and J. D. Schaffer, Eds. Los Alamitos, CA: IEEE Computer Soc., 1992, pp. 55–74.
- [23] H. Kitano, "Designing neural network using genetic algorithm with graph generation system", *Complex Systems* 1990, vol 4, pp.461-476.
- [24] Y. Liu and X. Yao "Evolutionary design of artificial neural networks with different nodes," in *Proceedings of IEEE International Conference on Evolutionary Computation (ICEC'96)*, Nagoya, Japan, 1996, pp. 670–675.
- [25] N. Dodd "Optimization of neural network structures using genetic techniques" *Internal Report RIPREP/1000/63/89*, Royal Signals and Radar Establishment, Malvern, UK, 1989

- [26] G. Miller, P. Todd, and S. Hegde “Designing neural networks using genetic algorithms” in *Proceedings of the Third Conference on Genetic Algorithms and their Applications*, San Mateo, CA. Morgan Kaufmann, 1989
- [27] A. Lindenmayer “Mathematical models for cellular interaction in development I, II” *Journal of theoretical biology* 1968, vol 18, pp280-315
- [28] G. D. Wilensky, and N. Manukian “The projection neural network” in *Proceedings of the International Joint Conference on Neural Networks 1992*, vol. 2, pp. 358-367.
- [29] M. Mandischer, “Representation and evolution of neural networks” R. F. Albrecht, C. R. Reeves, and U. C. Steele, editors, *Artificial Neural Nets and Genetic Algorithms*, Springer Verlag, pp. 643-649, 1993.
- [30] S. A. Harp, T. Samad, and A. Guha, “Designing application-specific neural networks using the genetic algorithm”, in *Proceedings of IEEE Conference on Neural Information Processing Systems, San Matco, 1990*, vol.2, pp. 477-454.
- [31] D. W. Opitz and J. W. Shavlik, “Connectionist Theory Refinement: Genetically Searching the Space of Network Topologies”, *Journal of Artificial Intelligence Research* 1997, vol. 6, pp.177-209.
- [32] J. C. F. Pujol and R. Poli, “Dual Network Representation applied to the Evolution of Neural Controllers”, *Seventh Annual Conference on Evolutionary Programming*, LNCS, vol. 1447, pp. 637-646, San Diego, March 25-27, 1998. Springer-Verlag.
- [33] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies”, *Evolutionary Computation* 10(2):99-127, 2002.
- [34] R. Poli, “Evolution of Graph-like Programs with Parallel Distributed Genetic Programming”, in *Proceedings of the Seventh International Conference on Genetic Algorithms,, East Lansing, July, 1997*
- [35] N.J. Radcliffe, “Genetic set recombination and its application to neural network topology optimization”, *Neural Computing and Applications*, 1(1), pp.67-90.
- [36] W. B. Langdon, “Size fair and homologous tree genetic programming crossovers,” *Genetic Programming and Evolvable Machines*, vol.1, no.1/2, pp.95-119, Apr 2000.
- [37] P. D'haeseleer, “Context preserving crossover in genetic programming,” in *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pp. 256 – 261, 27-29 June 1994

- [38] E. E. Korkmaz, G. Ucoluk, "A Controlled Genetic Programming Approach for the Deceptive Domain," *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, vol.34 , no.4, pp. 1730 – 1742, 2004
- [39] M. A. Lones, A. M. Tyrrell, "Crossover and bloat in the functionality model of enzyme genetic programming," in *Proceedings of the 2002 Congress on Evolutionary Computation*, vol.1 , pp. 986 – 991, 12-17 May 2002
- [40] M. I. Heywood, A.N. Zincir-Heywood, , "Dynamic page based crossover in linear genetic programming," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 32(3) , pp. 380 – 388, June 2002
- [41] M. J. Heywood, A. N. Zincir-Heywood," Page-based linear genetic programming," *IEEE International Conference on Systems, Man, and Cybernetics*, vol.5, pp.3823 – 3828, 8-11 Oct. 2000
- [42] G. C. Wilson, M. I. Heywood, "Crossover context in page-based linear genetic programming," *IEEE Canadian Conference on Electrical and Computer Engineering*, vol. 2 , pp. 809 – 814, 12-15 May 2002
- [43] M. D. Terrio, M. I. Heywood, "Directing crossover for reduction of bloat in GP," *IEEE Canadian Conference on Electrical and Computer Engineering*, vol 2 , pp.1111 – 1115, 12-15 May 2002
- [44] T.Ito, H. Iba, S. Sato, "Depth-dependent crossover for genetic programming," in *Proceedings of IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on Evolutionary Computation.,* pp. 775 – 780, 4-9 May 1998
- [45] A. Niimi, E. Tazaki, "Extended genetic programming using reinforcement learning operation" in *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, vol.5, pp. 596 - 600, 12-15 Oct. 1999
- [46] D.A. Augusto, H. J. C. Barbosa, "Symbolic regression via genetic programming," in *Proceeding in Sixth Brazilian Symposium on Neural Networks*, pp. 173 – 178, 22-25 Nov. 2000
- [47] M. Yanagiya, "Efficient genetic programming based on binary decision diagrams," *IEEE International Conference on Evolutionary Computation*, vol.1, pp. 234, 29 Nov.-1 Dec. 1995

- [48] J. Page, R. Poli, W. B. Langdon, "Smooth uniform crossover with smooth point mutation in genetic programming: A preliminary study," in R. Poli, P. Nordin, W. B. Langdon, and T. C. Fogarty, editors, in *Proceedings of Genetic Programming, EuroGP' 99*, LNCS, Goteborg, Sweden, 26-27 May 1999. Springer-Verlag.
- [49] R. Poli, N.F. McPhee, "Exact schema theory for GP and variable-length GAs with homologous crossover", in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 104-111, San Francisco, California, USA, 2001.
- [50] H. Bersini, "The immune and the chemical crossover," *IEEE Transactions on Evolutionary Computation*, 6(3), pp. 306 – 313, June 2002
- [51] C. Gathercole, P. Ross, "An adverse interaction between crossover and restricted tree depth in genetic programming," in *Proceedings of the First Annual Conference in Genetic Programming*. MIT Press. Stanford University, CA, USA, pp.291 – 296, 1996.
- [52] R. Poli, W. B. Langdon, "An analysis of the MAX problem in genetic programming," in *Proceedings of the Second Annual Conference in Genetic Programming*. Morgan Kaufmann. University of Wisconsin, Madison, Wisconsin, USA. pp. 222 – 230.
- [53] R. Poli, W. B. Langdon, "On the search properties of different crossover operators in genetic programming," in *Proceedings of the Third Annual Conference in Genetic Programming*. Morgan Kaufmann. University of Wisconsin, Madison, Wisconsin, USA. pp.293 – 301, 1998.
- [54] D. Chen, T. Aoki, N. Homma, T. Terasaki, T. Higuchi, "Graph-Based Evolutionary Design of Arithmetic Circuit," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp86-100, Feb 2000.
- [55] R.R. Behrens, "Art, Design and Gestalt Theory", *Leonardo: Journal of the International Society of Arts, Sciences, and Technology*, vol. 31, Number 4, pp299-304, August/September 1998.
- [56] S. Umeyama, "An eigendecomposition approach to weighted graph matching problems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, 1988.

- [57] S. Raghavan, H. Garcia-Molina, "Representing Web Graphs," in *Proceedings of 19th International Conference on Data Engineering, 2003*. pp.405 – 416, 5-8 March 2003.
- [58] J.R. Ullmann, "An Algorithm for Subgraph Isomorphism," *Journal of the Association for Computing Machinery*, vol. 23, pp. 31-42, 1976.
- [59] G. Schneider, O. Clement-Chomienne, L. Hilfiger, P. Schneider, S. Kirsch, H-J. Boehm, and W. Neidhart, "Virtual screening for bioactive molecules by evolutionary De Novo design", *Angewandte Chemie*, vol.39, pp.4130-4133, 2000.
- [60] D. Levine, M. Facello, P. Hallstrom, G. Reeder, B. Walenz, F. Stevens, "STALK: an evolutionary system for virtual molecular docking," *IEEE Computer Science Engineering*, vol.4, pp.55-65, 1997.
- [61] J. Wang, T. Hou, L. Chen, X. Xu,"Conformational analysis of peptides using Monte Carlo simulations combined with the genetic algorithm", *Chemometrics and Intelligent Laboratory Systems*, vol.45, no.1, pp.347-351, 1999.
- [62] J.C. Meza, M.L. Martinez, "On the use of direct search methods for the molecular conformation problem", *Journal of Computational Chemistry*, vol.15, no.6, pp.627-632, 1994.
- [63] H.Z. Xu, X.Y. Wei, M.S. Xu, "Schema analysis of multi-point crossover in genetic algorithm", in *Proceedings of the 3<sup>rd</sup> World Congress on Intelligent Control and Automation*, Hefei, P.R.China, June-July 2000.
- [64] Q.N. Hu, Y.Z. Liang, K.T. Fang, "The matrix expression, topological index and atomic attribute of molecular topological structure", *Journal of Data Science*, vol.1, pp.361-389, 2003.
- [65] O. Ivanciuc, M.V. Diudea, "Building-block computation of Ivanciuc-Balaban indices for the virtual screening of combinatorial libraries", *Internet Electronic Journal of Molecular Design*, vol.1, no. 1, pp.1-9, Jan2002.
- [66] I. Gutman, O.E. Polansky, "Mathematical concepts in organic chemistry", Springer-Verlag, Berlin, Heidelberg, New York, 1986.
- [67] K.A. De Jong, W. Spears, "On the virtues of parameterized uniform crossover", in *Proceedings of the 4<sup>th</sup> International Conference on Genetic Algorithms*, San Mateo: Morgan Kaufmann Publishers, pp 230-236, 1991.



- [68] K.A. De Jong, W. Spears, "An analysis of the interacting roles of population size and crossover in genetic algorithms", in *Proceedings of the First Int'l Conf. on Parallel Problem Solving from Nature*, Dortmund, Germany, October 1990.
- [69] C. Rücker, G. Rücker, M. Meringer, "Exploring the limits of graph invariant- and spectrum-based discrimination (sub)structures", *Journal of Chemical Information and Computer Sciences*, vol. 42, pp. 640-650, 2002.
- [70] R. Poli and W. B. Langdon, "Schema Theory for Genetic Programming with One-point Crossover and Point Mutation", *Evolutionary Computation Journal*, 6(3): 231-252, 1998.
- [71] T. Baeck, D.B. Fogel, Z. Michalewicz, "Handbook of Evolutionary Computation", *IOP Publishing*, Bristol, 1997.
- [72] D. Doval and D. O'Mahony, "Overlay networks – a scalable alternative for P2P", *IEEE Computer Society*, Jul-Aug 2003.
- [73] Y.H. Chu, S.G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast", *IEEE Journal on Selected Areas in Communications*, vol.20, no.8, Oct 2002.
- [74] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the Gnutella network: properties of large-scale peer-to-peer systems and implications for system design", *IEEE Internet Computing*, 2002.
- [75] L. Xiao, Y. Liu, and L.M. Ni, "Improving unstructured peer-to-peer systems by adaptive connection establishment", *IEEE Transactions on Computers*, vol.54, no.9, Sept2005.
- [76] Y. Liu, L. Xiao, and A.H. Esfahanian, L. M. Ni, "Approaching optimal peer-to-peer overlays", in *Proceedings of the 13<sup>th</sup> IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2005.
- [77] L.M. Ni and Y. Liu, "Efficient peer-to-peer overlay construction", in *Proceedings of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business*, 2004.
- [78] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P systems scalable", in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003.

- [79] C. Wang, and B. Li, "Peer-to-Peer overlay networks: a survey", Department of Computer Science, The Hong Kong University of Science and Technology, 2002.
- [80] S. Ren, L. Guo, S. Jiang, and X. Zhang, "SAT-Match: a self adaptive topology matching method to achieve low lookup latency in structured P2P overlay networks", in *Proceedings of the 18<sup>th</sup> International Parallel and Distributed Processing Symposium*, 2004.
- [81] Y. Liu, Z. Zhuang, L. Xiao, and L. Ni, "AOTO: adaptive overlay topology optimization in unstructures P2P systems", in *Proceedings of IEEE GLOBECOM 2003, San Francisco, USA, December 1-5, 2003*.
- [82] Z. Li and P. Mohapatra, "QRON: QoS-aware routing in overlay networks", *IEEE Journal on Selected Areas in Communications*, vol.22, no.1, Jan2004.
- [83] Z. Duan, Z.L. Zhang, and Y.T. Hou, "Service overlay networks: SLAs, QoS and bandwidth provisioning", in *Proceedings of the 10<sup>th</sup> IEEE International Conference on Network Protocols*, 2002.
- [84] S. L. Vieira and J'org Liebeherr, "Topology design for service overlay networks with bandwidth guarantees.," in *IWQoS*, 2004, pp. 211–220.
- [85] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, "Introduction to Algorithms", Second Edition, MIT Press 2001.
- [86] J.R. Kim and M. Gen, "Genetic algorithm for solving bicriteria network topology design problem," in *Proceedings of the 1999 Congress on Evolutionary Computing*, vol.33, 6-9 July 1999.
- [87] J. Gottlieb and L. Paulmann, "Genetic algorithms for fixed charge Transportation problem," in *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pp.330-335, IEEE Press, 1998.
- [88] G. R. Raidl, "An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem," in *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*, pp.104-111, IEEE Press, 2000.
- [89] J. Gottlieb, B.A. Julstrom, G.R. Raidl G.R., F. Rothlauf , "Prüfer numbers: a poor representation of spanning trees for evolutionary search," in *Proceedings of Genetic and Evolutionary Computation Conference*, San Francisco, CA, pp. 343–350, July 2001.

- [90] C. C. Palmer and A. Kershenbaum, "Representing trees in genetic algorithms," in *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 379–384. IEEE Press, 1994.
- [91] Dengiz, B., F. Altıparmak, and A. E. Smith, "Efficient optimization of all-terminal reliable networks using evolutionary approach", *IEEE Transaction on Reliability*, Vo1.46, No.1, pp.18-26, 1997.
- [92] D. Zhu, M. Gritter, D.R. Cheriton, "Feedback based routing," in *Proceedings of Workshop on Hot Topics in Networks (Princeton, New Jersey, Oct. 2002)*, pp.71-76, 2002.
- [93] P. Zhu, R.C. Wilson, "A study of graph spectra for comparing graphs", *Pattern Recognition*, vol.41(9), pp.2833-2841, Sept.2008.
- [94] E. R. van Dam, W. Haemers, "Which graphs are determined by their spectrum?", *Linear Algebra and its Applications*, vol.373, pp. 241–272, 2003
- [95] M. Lepovič, I. Gutman, "No starlike trees are cospectral", *Discrete Mathematics*, vol.242, pp. 291-295, 2002.
- [96] J.J. Michalek, R. Choudhary and P.Y. Papalambros, "Architectural layout design optimization", *Engineering Optimization*, vol.34(5), pp. 461–484, 2002.
- [97] J.S. Gero, and V.A. Kazakov, "Evolving design genes in space layout planning problems", *Artificial Intelligence in Engineering*, vol. 12(3), pp. 163-176, 1998.
- [98] M. Rosenman, "The generation of form using an evolutionary approach", in *Evolutionary Algorithms in Engineering Applications*, D. Dasgupta and Michalewicz, Eds, Springer-Verlag, pp. 69-86, 1997.
- [99] T. Schnier, and J.S. Gero, "Learning genetic representations as alternative to hand-coded shape grammars", in J. S. Gero and F. Sudweeks (Eds), *Artificial Intelligence in Design*, Kluwer, Dordrecht, pp.39-57, 1996.
- [100] J.S. Gero, V. Kazakov and T. Schnier, "Genetic engineering and design problems", in D. Dasgupta and Z. Michalewicz (Eds), *Evolutionary Algorithms in Engineering Applications*, Springer Verlag, Berlin, pp.47-68, 1997.
- [101] V. Venkatasubramanian, K. Chan, and J.M. Caruthers, "Evolutionary design of molecules with desired properties using the genetic algorithm", *Journal of Chemical Information and Computer Sciences*, vol.35, pp.188-195, 1995.

- [102] R. Nachbar, "Molecular evolution: a hierarchical representation for chemical topology and its automated manipulation", in *Proceedings of the Third Annual Genetic Programming Conference*, University of Wisconsin, Madison, Wisconsin, pp. 246-253, July 1998.
- [103] J. Devillers, "Designing Molecules with Specific Properties from Intercommunicating Hybrid Systems", *Journal of Chemical Information and Computer Sciences*, vol.36, pp.1061-1066, 1996.
- [104] J. Devillers, C. Putavy, "Designing biodegradable molecules from the combined use of back propagation neural network and a genetic algorithm", *Genetic Algorithms in Molecular Modelling*, Academic Press Limited, N.Y., pp.303-314, 1996.
- [105] R. Gani, "CAMD: Computer aided molecular design – examples of applications", CAPEC, Department of Chemical Engineering, Technical University of Denmark, DK-2800, Denmark, Nov 2002.
- [106] L.E.K. Achenie, R. Gani, V. Venkatasubramanian, "Computer Aided Molecular Design: Theory and Practice", Elsevier Press, 2003.
- [107] D.E. Clark, "Evolutionary algorithms in computer-aided design: a review of current applications and a look to the future", *Rational Design: Novel Methodology and Practical Applications*, ACS Symposium Series, vol.719, pp.255-270, American Chemical Society, 1999
- [108] D.K. Gehlhaar, G.M. Verkhivker, P.A. Rejto, C.J. Sherman, D.B. Fogel, L.J. Fogel, S.T. Freer, "Molecular recognition of the inhibitor AG-1343 by HIV-1 protease: conformationally flexible docking by evolutionary programming. *Chemistry and Biology*, vol.2, pp.317-324, 1995.
- [109] I.D. Kuntz, "Structure-based strategies for drug design and discovery", *Science*, vol.257, pp.1078-1082, 1992.
- [110] J.C. Meza, T.D. Plantenga, R.S. Judson, "Novel Applications of optimization to molecule design", *Institute for Mathematics and Its Applications*, vol. 94, pp.73, 1997.

- [111] R.S Judson, M.E. Colvin, J.C. Meza, A. Huffer, D. Gutierrez, "Do intelligent configuration search techniques outperform random search for large molecules ?", *International Journal of Quantum Chemistry*, vol.44, pp.277-290, 1992.
- [112] S.R. Wilson, W. Cui, "Applications of simulated annealing to peptides," *Biopolymers*, vol.29, pp.225-235, 1990.
- [113] C.E. Chang, M.K. Gilson, "Tork: conformational analysis method for molecules and complexes," *Journal of Computational Chemistry*, vol.24, pp.1987-1998, 2003.
- [114] J. Devillers, "Genetic algorithms in molecular modeling", *Academic Press*, New York, 1996
- [115] T. Hurst, "Flexible 3D searching: directed tweak technique", *Journal of Chemistry Information and Computer Science*, vol.34, pp.190-196, 1999.
- [116] G. Schneider, M.L. Lee, M. Stahl, P. Schneider, "De novo design of molecular architectures by evolutionary assembly of drug-derived building blocks", *Journal of Computer-Aided Design*, vol.14, no.5, pp.487-494, July, 2004.
- [117] P. Tuffery, C. Etchebest, S. Hazout, R. and Lavery, "A New Approach to the Rapid Determination of Protein Side Chain Conformations", *Journal of Biomolecular Structure and Dynamics*, vol.8, pp.1267-1289, 1991
- [118] M.J.J. Blommers, C.B. Lucasius, G. Kateman, and R. Kaptein, "Conformational analysis of a dinucleotide photodimer with the aid of the genetic algorithm", *Biopolymers* 1992, vol.32, pp.45-52, 1992.
- [119] A.H.C. van Kampen, and L.M.C. Buydens, "The ineffectiveness of recombination in a genetic algorithm for the structure elucidation of a heptapeptide in torsion angle space. A Comparison to Simulated Annealing", *Chemometrics and Intelligent Laboratory Systems*, vol.36, pp.141-152, 1997.
- [120] R.C. Glen, and A.W.R. Payne, "A genetic algorithm for the automated generation of molecules within constraints", *Journal of Computer-Aided Molecular Design*, vol.9, pp.181-202, 1995.
- [121] Y. Liu, L. Xiao, L. M. Ni, "Building a scalable bipartite P2P overlay network," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 18, no. 9, pp.1296-1306, 2007.

- [122] A. Shaikh, Kang Shin, "Destination-driven routing for low cost multicast", *IEEE Journal on Selected Areas in Communications*, vol.15, no.3, pp.373-381, April 2007.
- [123] S. Khuller, B. Raghavachari, N. Young, "Balancing minimum spanning trees and shortest path trees", *Algorithmica*, Springer N.Y., vol.14, no.4, pp.305-321, Oct.2008.
- [124] C.M. Lin, Y.T. Tsai, C.Y. Tang, "Balancing minimum spanning trees and multi-source minimum routing cost spanning trees on metric graphs", *Information Processing Letters*, vol.99, no.2, pp.64-67, July2006
- [125] Y. Zhang, N. Duffield, V. Paxson, S. Shenker, "On the Constancy of Internet Path Properties," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 2001.
- [126] S. Saroiu, P.K. Gummadi, S.D.Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems", in *Proceedings of Multimedia Computing and Networking (MMCN'02)*, San Jose, CA, Jan. 2002.
- [127] M. Faloutsos, P. Faloutsos, C. Faloutsos, "On power-law relationships of the Internet topology", in *Proceedings of ACM SIGCOMM'1999*, pp. 251–262, Aug./Sept. 1999.
- [128] Y. Liu, X. Liu, L. Xiao, L. M. Ni, X. Zhang, "Location-aware topology matching in P2P systems," in *Proceedings of IEEE INFOCOM*, 2004.
- [129] Y. Liu, "A two-hop solution to solving topology mismatch", *IEEE Transactions on Parallel and Distributed Systems*, vol.19, no.11 pp.1591-1600, Nov 2008.
- [130] K.S. Seo, Z. Fan, J.J. Hu, E.D. Goodman, R.C. Rosenberg, "Toward a unified and automated design methodology for multi-domain dynamic systems using bond graphs and genetic programming : Computational intelligence in mechatronic systems", *Mechatronics*, vol.13, np.8-9, pp.851-885, 2001.
- [131] M. Fischer, H. Gall, "EvoGraph: A Lightweight Approach to Evolutionary and Structural Analysis of Large Software Systems", *Working Conference on Reverse Engineering*, 2006. vol.13, pp.179 – 188, Oct. 2006.
- [132] M. McGlohon, C. Faloutsos, "ADAGE: a software package for analyzing graph evolution", Carnegie Mellon University, May 2007.

- [133] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolutionary Programs*, Springer, Berlin, 1996.
- [134] L. J. Fogel, A. T. Owens, M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, John Wiley, New York, 1966.
- [135] J. H. Holland, "Outline for a logical theory of adaptive systems," *Journal of the Association for Computing Machinery*, vol. 3, pp. 297–314, 1962.
- [136] K. A. De Jong, "Are genetic algorithms function optimizers?" in *Parallel Problem Solving from Nature 2*. pp. 3–13, Amsterdam, The Netherlands: Elsevier, 1992,.
- [137] L. J. Fogel, "On the organization of intellect," Ph.D. dissertation, University of California, Los Angeles, 1964.
- [138] H. P. Schwefel, *Evolution and Optimum Seeking*. New York: Wiley, 1995 (Sixth-Generation Computer Technology Series).
- [139] J. H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, 1992.
- [140] R. Raidl, "An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem," in *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*, pp.104-111, IEEE Press, 2000.
- [141] F. Routhlauf, D. Goldberg, and A. Heinzl, "Network random key -A tree network representation scheme for genetic and evolutionary algorithms," *Evolutionary Computation*, vol. 10, pp. 75–97, 2002.