THE UNIVERSITY OF WARWICK

Network Coding Via Evolutionary Algorithms

Lalith Karunarathne

Sponsoring Organisation	EPSRC
Address	Polaris House North Star Avenue Swindon SN2 1ET

NETWORK CODING VIA EVOLUTIONARY ALGORITHMS

By

Lalith Karunarathne

This thesis is submitted in partial fulfilment of the requirements for the award of the degree
Doctor of Philosophy in Engineering

University of Warwick, School of Engineering
November 2012

Sponsoring Organisation	EPSRC
Address	Polaris House North Star Avenue Swindon SN2 1ET

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank the Engineering and Physics Science Research Council (EPSRC) for providing me the necessary financial support. Special thanks to my supervisor Dr. Mark Leeson for his invaluable guidance, encouragement and help in many ways; my second supervisor Prof. Evor Hines for his faithful support; my panel members Prof. Roger Green and Dr. Christos Mias for being wonderful mentors; research degrees secretary Kerrie Hatton for being a friendly coordinator; my family for their loyal support, and except whose help and support this thesis would still be very far from completion.

DECLARATION

This is to certify that I am responsible for the work submitted in this thesis, that the original work is my own except as specified in acknowledgments or in footnotes, and that neither the thesis nor the original work contained therein has been submitted to this or any other institution for a higher degree.

As part of the work carried out, the following paper was submitted (under review):

L.P. Karunarathne, M.S. Leeson and E.L. Hines, "Evolutionary Minimisation of Network Coding Resources", *submitted to Computer Communications,* February 2013.

The manuscript is available in Appendix C.

ABSTRACT

Network coding (NC) is a relatively recent novel technique that generalises network operation beyond traditional *store-and-forward* routing, allowing intermediate nodes to combine independent data streams linearly. The rapid integration of bandwidth-hungry applications such as video conferencing and HDTV means that NC is a decisive future network technology.

NC is gaining popularity since it offers significant benefits, such as throughput gain, robustness, adaptability and resilience. However, it does this at a potential complexity cost in terms of both *operational* complexity and *set-up* complexity. This is particularly true of network code construction.

Most NC problems related to these complexities are classified as *non deterministic polynomial hard* (NP-hard) and an evolutionary approach is essential to solve them in polynomial time. This research concentrates on the multicast scenario, particularly: (a) network code construction with optimum network and coding resources; (b) optimising network coding resources; (c) optimising network security with a cost criterion (to combat the unintentionally introduced Byzantine modification security issue).

The proposed solution identifies minimal configurations for the source to deliver its multicast traffic whilst allowing intermediate nodes only to perform forwarding and coding. In the method, a preliminary process first provides unevaluated individuals to a search space that it creates using two generic algorithms (augmenting path and linear disjoint path. An initial population is then formed by randomly picking individuals in the search space. Finally, the Multi-objective Genetic algorithm (MOGA) and Vector evaluated Genetic algorithm (VEGA) approaches search the population to identify minimal configurations. Genetic operators (crossover, mutation) contribute to include optimum features (e.g. lower cost, lower coding resources) into feasible minimal configurations. A fitness assignment and individual evaluation process is performed to identify the feasible minimal configurations.

Simulations performed on randomly generated acyclic networks are used to quantify the performance of MOGA and VEGA.

Thesis Supervisor:	Mark Leeson
Title:	Associate Professor (Reader)

ACRONYMS / ABBREVIATIONS USED

NC: Network coding

HDTV: High-definition television

$\mathbb{F}_q$: Finite field with q element

GAs: Genetic algorithms

NP: Nondeterministic polynomial time

$G(V,E)$: A directed graph with node set V and edge set E

$G' \in G$: $G'(V',E')$ is a subgraph of G if $V' \subseteq V$ and $E' \subseteq E$

(v_i, v_j) : Link or edge, $v_i, v_j \in V$; $(v_i, v_j) \in E$

$\left\langle v_0, v_1, \ldots. v_i, v_j \ldots.. v_k \right\rangle$: A path from v_0 to v_k

$|V|, |E|$:A number of nodes, and a number of edges in the network $G(V,E)$

XOR, $\oplus$: Exclusive OR

1 INTRODUCTION

Network coding (NC) is an elegant technique introduced to improve the efficiency of transmission in bandwidth-hungry applications such as telecommuting, video conferencing, e-learning, HDTV and a host of other business applications in a multicast scenario. With the rapid integration of these applications, NC is expected to be a critical technology for future network solutions. Moreover the network coding technology is populating very diverse dimensions of communication networks, because it offers significant benefits. These include throughput gain, wireless resources savings, security enhancements, complexity suppression, robustness and adaptability, and resilience to link failures. NC deployment is challenged by a number of factors relating, *inter alia*, to code construction, resource usage and security. This thesis concerns network coding resources, network code construction and secure network coding with a cost criterion in a multicast scenario. This versatile concept appeared in the network environment at the turn of the millennium, and researchers in a diversity of fields such as computer science, mathematics and engineering were attracted with a significant interest. Research to date has often concentrated its efforts on overcoming these challenges which are mostly categorised as NP-hard problems. Instead of tackling their complexity, efforts have focused on the discovery of good solutions via evolutionary algorithms. This work provides solutions to the same kind of problems using an evolutionary algorithm based on genetic algorithms (GAs). The formulated problems comprise multiple objectives, therefore a traditional generic single-objective GAs are modified to find a set of multiple non-dominated solutions in a single run.

1.1 EVOLUTIONARY APPROACHES TO NETWORK CODES CONSTRUCTION

Network code construction is one of the major challenges in the multicast scenario. Fragouli and Soljanin discuss two common initial procedures to construct network codes for multicasting [1]. Given a multicast instance $\{G=(V,E),S,R\}$, the first common steps are:

1. Find h edge-disjoint paths $\{(S_i,R_j),1\le i\le h;1\le j\le N\}$ from the source to the receivers, the associated graph $G'=\bigcup_{1\le j\le N}^{1\le i\le h}(S_i,R_j)$ with the set of coding points C, and
2. Find the associated minimal configuration.

The identification of the minimal configuration with a minimum number of coding points is NP-hard [1]. The code construction method of the proposed solution in section 4.2 of this thesis intends to identify the minimal configuration with optimal network and coding resources which is also defined as the NP-hard problem. The proposed solution, based on a GA, accepts the challenge of solving this and quickly identifies a solution instead of tackling the NP-hard problem.

1.2 EVOLUTIONARY APPROACHES TO NETWORK CODING RESOURCE MINIMISATION

Network coding resources, their exhaustion in the multicast scenario and evolutionary approaches to minimise them are briefly discussed here. Fundamentally, the coding nodes are enriched in terms of buffer memory, computational capability and operating power, and these additional abilities are defined as the coding resources [1], [2]. These resources are rapidly consumed and ultimately exhausted by computational complexity, packet delay, congestion, packet misrouting and so forth. The packet delay, congestion and packet misrouting contribute to cause synchronising errors at the coding nodes and decoding errors at the sinks. The network coding resources for multicasting are comprehensively discussed by Fragouli and Soljanin [1], who describe the major complexity components are as *Set-up complexity* and *Operational complexity.*

Before NC was introduced to the communication network scenario, network nodes only performed the packet routing and duplication functions. In Figure 1-1(a) sample network, nodes *A*, *B*, and *D* only perform the packet routing and duplication functions, and packet $'a'$, $'b'$ and $'a \oplus b'$ are duplicated and routed by nodes *A*, *B*, and *D* consecutively. Node *C* is distinguishable from the others because it is functionally integrated by the NC technique. In Figure 1-1(a), packet $'a'$and $'b'$ are typically asynchronously reaching node *C*, and packet $'a'$is advanced by time T compared to$'b'$. During this time period, the input buffer memory allocates storage for packet $'a'$and operating power is consumed to maintain the buffer. Furthermore node *C* allocates its computational power to a simple exclusive-OR (XOR) binary operation to form packet$'a \oplus b'$.

Set-up complexity and *Operational complexity* and their related factors (e.g. a number of coding nodes, a number of input links per coding nodes, etc) are concerned in this research, and the proposed GA-based solution is discussed in section 5.3. The former denotes the complexity of designing the network coding scheme, which includes selecting the paths through the information flows and determining the operations (coding, forwarding etc.) that the nodes of the network perform. In Figure 1-1(b), where source *S* wishes to transmit data at rate 2 to the 3-leaf nodes *A*, *B* and *G*, and the paths should be selected through information flows and the nodes operations of selecting paths thus be determined. Either node *C* or node *E* should perform the coding operation to achieve the multicast rate 2. If both nodes *C* and *E* perform the coding operation, they are unnecessary to achieve the multicast rate 2 and cause decoding error at sink $t_{2,}$ consequently the operations contribute to exhaustion of the coding resources. The latter encompasses the running cost of using network coding, that is the amount of computational and network resources required per information unit successfully delivered.

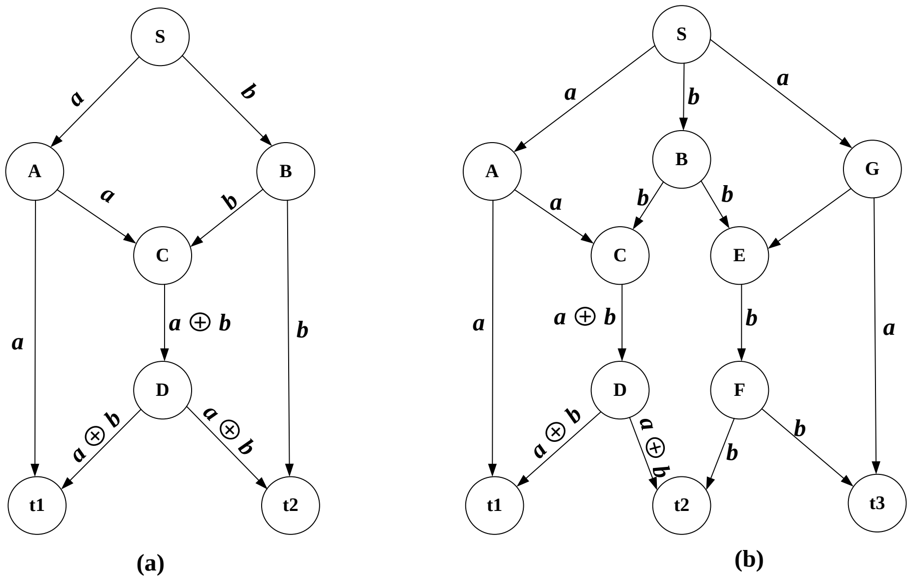


Figure 1-1: Sample networks

The proposed solution in section 5.3 identifies the minimal configuration with optimised network coding resources which is NP-hard. This minimal configuration is the ideal solution to suppress both complexities.

1.3 EVOLUTIONARY APPROACH FOR SECURE NETWORK CODING WITH A COST CRITERION

This is a first attempt to investigate jointly network cost, coding cost, wiretapper adversaries and Byzantine modification in the multicast scenario. The cost calculation is considered as a basic function of resource (network resource and coding resource) allocation for a unit packet successfully delivered from the source to a set of sinks during a unit time period. NC is an elegant technique to protect (i.e. without additional security mechanisms) multicast data naturally against wire tappers. However, it not only offers benefits but also it unintentionally allows a fatal error which is Byzantine modification. A malicious node usually pretends to forward packets from source to sink. Since network coding makes the coded packets at the routers, a single corrupted packet can cause a fatal disruption to the decoding operations at the sinks. Moreover, uncoded packets are not protected except by costly randomness. Protecting the source messages from wiretappers via randomness is effective but contributes to exhaustion of the resources and consequently they affect a cost criterion. However the transmission in the network has to be randomised because otherwise a channel output would be either a function depending on the messages, or simply a constant.

The proposed solution in chapter 6 identifies low cost (network cost and coding cost) minimal configurations $(G' \in G)$ in an adversary network, which is categorised as NP-hard; an evolutionary approach is essential to solve it. These G's are classified as *highly vulnerable* $G_H^{'}$s and *lower vulnerable* $G_L^{'}$s. The $G_L^{'}$s can only be protected from the wiretappers but they cannot be identified straightforwardly because the wiretapper adversaries cannot be detected. Simulation results show that multi-objective GA based techniques in the proposed solution have a high potential to identify the $G_L^{'}$s. Nevertheless, these may still not be perfectly protected because they may be comprised of malicious nodes. The network *G* is assumed to be error free and $G_L^{'}$s are examined for malicious nodes. Moreover links which deliver uncoded packets in $G_L^{'}$s are still threatened by the wiretappers, and they are perfectly protected by the proposed random coding and packet forwarding technique at the source without costly randomness.

1.4 CONTRIBUTIONS

Most problems in the NC concept are NP-hard and traditional solving methods (optimising, searching) have not been able to provide feasible solutions or tackling the problems. In this circumstance, this research introduces a new pathway to find a feasible solution instead of tackling the NP-hard problems. The aspiration is an identification of minimal configurations; sets of linear disjoint paths are combined by an evolutionary algorithm based on the GA. The approach introduces how diverse sets of parameters (e.g. network resources, coding resources etc.) are simultaneously optimised and include them into the minimal configurations identified.

Network code construction is a challenge and it is fatally affected by *setup* complexity (complexity of identifying paths through coding points and satisfying multicast demands such as the min-cut max-flow theorem). The proposed solution in chapter 4 is an excellent contribution to smooth out the complexity and provides benefits to develop practical network coding protocols.

NC resource is excessively consumed by *computational* complexity and the complexity is affected by a selected network coding scheme, a number of coding nodes which perform during the multicast transmission and a number of in-links for each coding node. Chapter 5 discusses the contribution via optimisation of the NC resource; the proposed method identifies the minimal configuration with optimal coding resources which is defined as NP-hard. The significant point is that the source is able to obtain explicit information about the coding operations and can select a limited size of finite field to its multicast transmission.

Network security is a vital topic in the cyber world. Most security mechanisms against adversaries entirely concerned with strengthening their security level instead of cost. Chapter 6 contributes to its proposed solution to develop a low cost secure network coding scheme against wiretapper adversaries and Byzantine modifications in the multicast transmission.

1.5 REFERENCES

[1] C. Fragouli and E. Soljanin, "Network Coding Fundamentals", *Foundation and Trends in Networking, Vol 2*, no.1, pp.1-133,2007.

[2] M. Kim, C. W. Ahn, M. Médard, and M. Effros, "On minimizing network coding resources: An evolutionary approach," Network Coding Workshop, 2006.

2 BACKGROUND TO THE RESEARCH

This chapter is allocated to providing an infrastructure to this thesis. Section 2.1 offers an overview of network coding. Section 2.2 discusses finite field operations. Section 2.3 shows the benefits of network coding. Section 2.4 includes the disadvantages of network coding. Section 2.5 is allocated to briefly discuss network coding applications. Section 2.6 consists of prior work in network coding relevant to this thesis.

2.1 OVERVIEW OF NETWORK CODING

All communication networks today make a basic assumption that *information is separate.* Thus, whether it is packets in the Internet, or signals in a phone network, if they originated from different sources, they are transmitted much in the same manner as cars on a transportation network of highways, or fluid through a network of pipes. That is, independent data streams may share network resources but the information itself is separate. Most network functions such as routing, data storage and error control are based on this assumption.

This assumption is broken by network coding as it allows intermediate nodes in the network to combine their input packets into one or more output packets. Network coding is best demonstrated through the famous butterfly network which is given in the seminal paper [1] of Ahlswede *et al*, shown in Figure 2-1. Each source produces one bit per unit time slot (unit rate sources).

If sink t_1 uses all the network resources by itself, it is able to receive both packets $'a'$ and $'b'$. Indeed, the packet $'a'$ could be routed by source S_a along the path

$\langle S_a, t_1\rangle$ and the packet $'b'$ by source S_b along $\langle S_b, C, D, t_1\rangle$, as depicted in Figure 2-1(a). Similarly, if sink t_2 consumes all network resources by itself, it could also receive both $'a'$ and $'b'$, as depicted in Figure 2-1(b).

Now assume that both sinks want to receive the information from sources S_a and S_b simultaneously. If routers C and D only forward the packets they receive, the middle link (C, D) will be a bottleneck arising from the fact that only one packet (1bit) per unit time slot through may be sent via this edge. However, packets $'a'$ and $'b'$ are simultaneously sent to reach the sinks t_2 and t_1 consecutively.

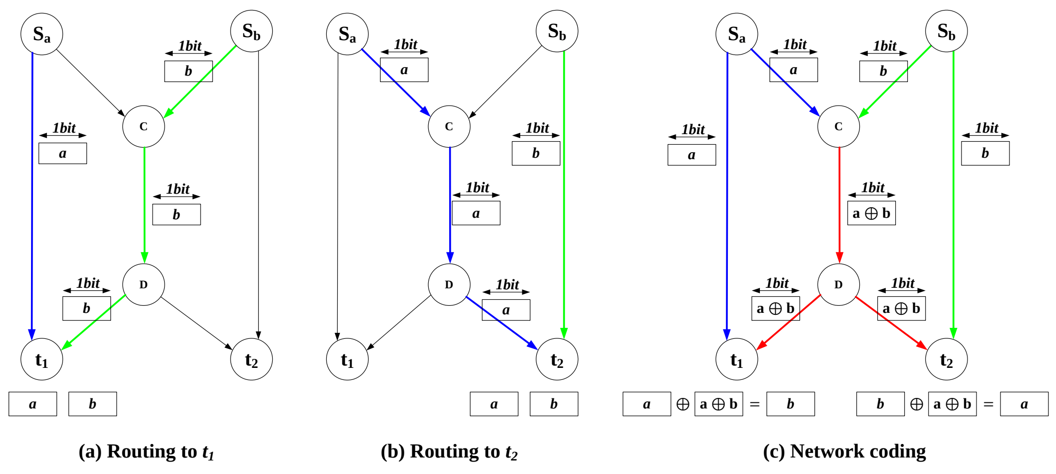


Figure 2-1: The Butterfly network. Sources S_a and S_b multicast their information to sinks t_1 and t_2.

Conventionally, information flow was considered similar to fluid through pipes, and independent information flows were distinct. Considering this approach node C would have to make a decision regarding forwarding the input packets: either use link (C, D) to send packet $'a'$, or use it to send packet $'b'$. Thus, when packet $'a'$ is chosen, sink t_1 will only receive $'a'$ and sink t_2 will receive both $'a'$ and $'b'$, and vice versa when $'b'$ is chosen.

The simple but vital observation was made in the seminal work by Ahlswede *et. al.*[1] that intermediate nodes in the network are allowed not only to forward their incoming information streams but also process them prior to forwarding. In particular, node *C* is able to combine packets $'a'$ and $'b'$ using an XOR (binary addition over binary field) binary operation and create a third packet $'a \oplus b'$ (1bit) it can then send through link (C, D), as depicted in Figure 2-1(c). The sinks t_1 and t_2 receive packets $\{'a', 'a \oplus b'\}$ and $\{'b', 'a \oplus b'\}$ consecutively, and can easily solve to retrieve the packets $'a'$ and $'b'$.

The XOR operation in network coding may be replaced by linear network coding to allow for a much larger degree of flexibility in the way that packets can be combined. Thus, routers combine packets linearly instead of simply forwarding them to create coded packets. The encoding and decoding processes are briefly described in the following sections.

2.1.1 ENCODING

Assume that each packet consists of *L* bits. When the packets to be combined, if their sizes are not equal, the shorter ones are padded with trailing 0s. Each packet is represented as *k* consecutive bits of a symbol over the finite field $\mathbb{F}_2{}^k$; thus each packet is a vector of *L*/*k* symbols. The linear network coding allows intermediate nodes in the network to combine their incoming packets linearly over the finite field $\mathbb{F}_2{}^k$. The linear combination uses addition and multiplication over the finite field $\mathbb{F}_2{}^k$. For example when $k = 1$, then the finite field $\mathbb{F}_2{}^1 = \{0, 1\}$ has a one bit symbol and a field size of 2; when $k = 2$, the size 4 finite field as $\mathbb{F}_2{}^2 = \{00, 01, 10, 11\}$ is obtained, and each symbol has 2 bits.

The discussion is initiated with the standard framework. An acyclic graph $G(V,E)$ consists of unit capacity edges, a sender $S \in V$, and a set of receivers $t_1, t_2 \ldots t_N \in T$. The multicast capacity *h* is the minimum number of edges in any cut between the sender and a receiver, which implies that *h- unit rate sources are present.* Each edge $e \in E$ emanating from a node $v = In(e)$ $(v \in V)$ carries a symbol $y(e)$ that is a linear combination of the symbol $y(e')$ on the edges e' entering *v*, namely, $\sum_{e':out(e')=v} m_e(e')y(e')$. The *local encoding vector* $\mathbf{m}(e) = \left[m_e(e')\right]_{e':out(e')=v}$ represents the encoding function at node *v* along edge *e*. If *v* is the sender *S*, then to maintain uniformity of notation, virtual edges $e_1'.........e_h'$ entering *S*, carrying the *h* source symbols $y(e_i') = x_i, i = 1,......h$ are introduced. Thus by induction $y(e)$ on any edge $e \in E$ is a linear combination $y(e) = \sum_{i=1}^{h} g_i(e)x_i$ of the source symbols, where the *h* dimensional vector of coefficients $\overrightarrow{g(e)} = [g_1(e),.........g_h(e)]$ can be determined recursively by $\overrightarrow{g(e)} = \sum_{e':out(e')=v} m_e(e')g(e')$, where $g(e_i')$ on the artificial edge e_i' is initialised to the i^{th} unit vector. The vector $\overrightarrow{g(e)}$ is known as the *global encoding vector* along edge *e*. Any receiver $t_1, t_2....t_N \in T$ can receive the symbols $\left[y(e_1)\ y(e_2).........y(e_h)\right]^T$ along its *h* (or more) incoming edges $e_1 ....e_h$. Each receiver can obtain the source symbol $x_1, x_2.....x_h$ by solving the equations in Figure 2-2 and the matrix G_t of global encoding vectors $\overrightarrow{g(e)} = [g_1(e_1),.........g_h(e_h)]$ should be of *full rank* (Appendix A1) to solve the system equations.

$$\begin{bmatrix} y(e_1) \\ \vdots \\ y(e_h) \end{bmatrix} = \begin{bmatrix} g_1(e_1)\cdots g_h(e_1) \\ \vdots \quad \ddots \quad \vdots \\ g_1(e_h)\cdots\ g_h(e_h) \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_h \end{bmatrix} = \begin{bmatrix} \overrightarrow{g(e_1)} \\ \vdots \\ \overrightarrow{g(e_h)} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_h \end{bmatrix} = G_t \begin{bmatrix} x_1 \\ \vdots \\ x_h \end{bmatrix}$$

Figure 2-2: The system equation for linear network coding model.

The full rank stipulation of the matrix G_t can be satisfied with high probability if local encoding vectors $\mathbf{m}(e)$ are generated randomly, which is called *Random Linear Network Coding* and the symbols of $\mathbf{m}(e)$ lie in the finite field ($\mathbb{F}_q$) of sufficient size (q- sufficiently large). Jaggi *et al.*[2] showed that with the finite field ($\mathbb{F}_{2^{16}}$) having field size ($q = 2^{16}$) and a number of edges in the network that is at most $|E| = 2^8$, then the matrix G_t at any given receiver will have full rank with a probability of at least $1 - 2^{-8} = 0.996$.

In a packet network, the symbols $y(e)$ carried along an edge e can be grouped into packets. Thus the symbols $y(e)$ flowing on each edge e are packetized into vectors $Y(e) = y_1(e), y_2(e), ...y_\psi(e)$ of the appropriate length (depending on the field size), and now each of these *vectors* can be expressed as a linear combination $Y(e) = \sum_{e':out(e')=v} m_e(e')Y(e')$ of the vectors $Y(e')$ on the edges e' entering $v = In(e)$. Likewise, the source symbols x_i are packetized flowing into the sender on the artificial edges e_i' into vectors $X_i = [x_{i,1}, x_{i,2} ...., x_{i,\psi}]$ so that any receiver can recover (with high probability) the h source vectors $X_1, X_2 ....., X_h$ from any h received packets,

$$\begin{bmatrix} Y(e_1) \\ \vdots \\ Y(e_h) \end{bmatrix} = \begin{bmatrix} y_1(e_1) & y_2(e_1)\cdots & y_\psi(e_1) \\ \vdots & \vdots\quad\ddots & \vdots \\ y_1(e_h) & y_2(e_h)\cdots & y_\psi(e_h) \end{bmatrix} = G_t \begin{bmatrix} X_1 \\ \vdots \\ X_h \end{bmatrix} = G_t \begin{bmatrix} x_{1,1} & x_{1,2}\cdots & x_{1,\psi} \\ \vdots & \vdots\quad\ddots & \vdots \\ x_{h,1} & x_{h,2}\cdots & x_{h,\psi} \end{bmatrix}$$

Figure 2-3: The system equations for the packetized linear network coding model.

2.1.2 DECODING

Assume any sink or receiver has received the set $[\overrightarrow{g(e_1)}, Y(e_1)], \ldots\ldots..[\overrightarrow{g(e_m)}, Y(e_m)]$. To retrieve the original packets, it is necessary to solve the system equations in Figure 2-2. This is a linear system with m equations and h unknowns and the condition $m \geq h$ must be satisfied to have a chance of recovering all data, i.e. the number of received packets needs to be at least as large as the number of original packets. Conversely, the condition $m \geq h$ may not be satisfied as some of the combinations might be *linearly dependent* (Appendix A1).

In practice, decoding requires solving a set of linear equations, which can be accomplished efficiently using Gaussian elimination. Each sink or receiver node stores the encoded vectors $\vec{g}(e)$ it receives as well as corresponding packets $Y(e)$, row by row, in a so-called *decoding matrix*. Initially, the matrix is empty. When an encoded packet is received, it is inserted as the next row in the decoding matrix and Gaussian elimination is performed to transform it to an upper triangular matrix (Appendix A2). Figure 2-4 shows how Gaussian elimination progresses for the Figure 2-3 system equations.

A received packet is called *innovative* if it increases the rank of the matrix, i.e. the packet is linearly independent. If a packet is non-innovative, it is reduced to a row of 0s by Gaussian elimination and is ignored, i.e. the packet is linearly dependent. Instantly, the matrix consists of a row of the form $\{g_j^{'}(e_j) \mid y_1^{'}(e_j)\ y_2^{'}(e_j) \cdots y_{\psi}^{'}(e_j)\}$, where j is any row of the upper triangular matrix in Figure 2-4(b). The form is same as the bottom row of Figure 2-4(b) and the sink or receiver can obtain the original source packets $\{x_{h,1}\ x_{h,2}\ x_{h,3} \cdots\cdots x_{h,\psi-1}\ x_{h,\psi}\}$ by forming them as

$\{x_{h,1} \equiv g_h'(e_h)\, y_1'(e_h);\; x_{h,2} \equiv g_h'(e_h) y_2'(e_h) \cdots x_{h,\psi} \equiv g_h'(e_h) y_\psi'(e_h)\}$. Note that decoding does not need to be performed at all nodes of the network, but only at the receivers.

$$\begin{bmatrix} g_1(e_1)\cdots g_h(e_1) & | \; y_1(e_1) \; y_2(e_1)\cdots \; y_\psi(e_1) \\ g_1(e_2)\cdots g_h(e_2) & | \; y_1(e_2) \; y_2(e_2)\cdots \; y_\psi(e_2) \\ \vdots \;\ddots \quad \vdots & \\ g_1(e_h)\cdots g_h(e_h) & | \; y_1(e_h) \; y_2(e_h)\cdots \; y_\psi(e_h) \end{bmatrix} \begin{bmatrix} 1 & & 0 & x_{1,1} & x_{1,2}\cdots & x_{1,\psi} \\ 0 & 1 & 0 & x_{2,1} & x_{2,2}\cdots & x_{2,\psi} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & & 1 & x_{h,1} & x_{h,2}\cdots & x_{h,\psi} \end{bmatrix}$$

(a)

$$\begin{bmatrix} g_1'(e_1)\cdots\cdots\cdots g_h'(e_1) & | \; y_1'(e_1) \; y_2'(e_1)\cdots \; y_\psi'(e_1) \\ 0 \;\; g_2'(e_2)\cdots\cdots g_h'(e_2) & | \; y_1'(e_2) \; y_2'(e_2)\cdots y_\psi'(e_2) \\ \vdots \;\; \vdots \quad \ddots \quad \vdots & \\ 0 \;\; 0 \;\; 0 \;\; 0\cdots \;\; g_h'(e_h) & | \; y_1'(e_h) \; y_2'(e_h)\cdots y_\psi'(e_h) \end{bmatrix} \begin{bmatrix} 1 & 0 & & 0 & x_{1,1} & x_{1,2}\cdots & x_{1,\psi} \\ 0 & 1 & & 0 & x_{2,1} & x_{2,2}\cdots & x_{2,\psi} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & x_{h,1} & x_{h,2}\cdots & x_{h,\psi} \end{bmatrix}$$

(b)

Figure 2-4: (a) the system equations compose to the "augmented matrix equation"; (b) perform elementary row operations to put the augmented matrix into the upper triangular form

2.1.3 PRACTICAL ISSUES

Practical issues in encoding and decoding are basically related to the size of the decoding matrix and the size of the finite field. For practical purposes, both parameters have to be limited. This can be achieved by grouping packets into *generations,* and mandating that only packets in the same generation can be combined linearly [3]. Moreover the parameters are limited by deterministic network codes but it is more difficult with random network coding. The size of the generation significantly affects the performance of network coding and it is related to the size of the finite field. Typically, a small finite field increases the probability of non-innovative transmissions and reduces the performance.

2.2 FINITE FIELD OPERATIONS

Network coding requires operations in the *binary field* ($\mathbb{F}_q$, q= 2^k), i.e. operations on strings of *k-bits*, which is popular as the Galois field[1] *GF*(q), where q is a number of elements in the field. One way to construct $\mathbb{F}_{2^k}$ is to use a *polynomial basis representation.* Here, the elements of $\mathbb{F}_{2^k}$ are the binary polynomials (polynomial whose coefficients are in the field $\mathbb{F}_{2^1} = \{0,\ 1\}$ of degree at most $k-1$:

$$\mathbb{F}_{2^k} = \left\{ a_{k-1}z^{k-1} + a_{k-2}z^{k-2} + \ldots\ldots + a_2z^2 + a_1z^1 + a_0z^0 : a_i \in \{0,1\} \right\}$$

An irreducible binary polynomial $f(z)$ of degree *k* is chosen; the *irreducibility* of $f(z)$ means that it cannot be factorised as a product of binary polynomials each of degree less than *k*. The addition of field elements is the usual addition of polynomials, with coefficient arithmetic performed modulo 2 (Appendix A3.1). The multiplication of field elements is performed modulo the *reduction polynomial* $f(z)$, (Appendix A3). For any binary polynomial $a(z)$, $a(z) \bmod f(z)$ shall denote the unique remainder polynomial $r(z)$ of degree less than *k* obtained upon long division of $a(z)$ by $f(z)$; this operation is called *reduction modulo* $f(z)$, (Appendix A3).

[1] French mathematician Évariste Galois

2.3 THE BENEFITS OF NETWORK CODING

Network coding provides benefits in a wide variety of scenarios, such as static wired or wireless networks, ad-hoc mobile wireless networks, wireless sensor network and optical networks[10],[4]. These benefits facilitate improvements in network throughput and security, saving resources. Network coding offers robustness and adaptability to a traditional multicast routing network. Moreover the complexity of content distribution is minimised by network coding.

2.3.1 THROUGHPUT GAIN IN A STATIC ENVIRONMENT

In communication networks, such as Ethernet or packet radio, throughput is described as the average rate of successful message delivery over a communication channel. These channels are basically physical or logical links. The throughput is usually measured by the unit of bits or bytes per second (bits/sec or bps) or by the data packet per second. Maximum throughput in each channel is constrained by its channel capacity and network coding is a promising method to improve this.

The primary result [1] shows that network coding can increase the capacity of a network for multicast flows. Consider a network in Figure 2-1 (a) or (b), sinks t_1 and t_2 are interested in simultaneously receiving data from both sources S_a and S_b. Each sink needs all the network resources when only traditional network routing is employed but as illustrated in Figure 2-1 (c), network coding allows to both sinks to receive data from both sources.

Network coding may offer throughput benefits not only for multicast flow, but also for other traffic patterns such as unicast. Considering Figure 2-1 (a) further it is now assumed that source S_a needs to transmit to the sink t_2 and source S_b needs to transmits to sink t_1. With network coding the source can send at rate 1 (1bit/sec) to each receiver as opposed to only ½ (0.5 bit/sec) without.

In the multicast scenario, with network coding, the source can send at rate 2 (2bit/sec) to each receiver. But the source can maximally achieve the rate 1½ (1.5 bit/sec) to each receiver if a network uses a traditional multicast routing. The reason for this difference is, network coding allows the combining of two bits into one time slot (1 sec) at node *C* in Figure 2-1 (c).

2.3.2 SECURITY

Sending linear combinations of packets instead of uncoded data offers a natural way to take advantage of multipath diversity for security against wiretapping attacks. The wiretap network (shown in Figure 2-5 with admissible codes) consists of a communication network and a collection of subsets of wiretap channels. Any link is susceptible to wiretapper adversaries and the admissible codes protect a source message *m* from wiretappers. The source generates a random packet *k* and combines it with the message *m.* The packets *k+m* and *k-m* are encoded at node 'a_0', and the packet *k* encoded is forwarded to sink t_1 and t_3.The admissible codes allow legal users t_1, and t_3 to obtain *m* without any errors. Moreover the wiretapper cannot obtain information about the secure message by accessing any *1- channel*. Thus networks that only require protection against such simple attacks can obtain this without additional security mechanisms.

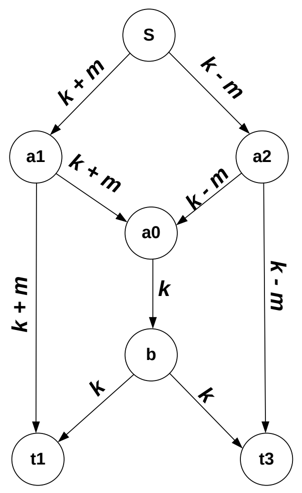


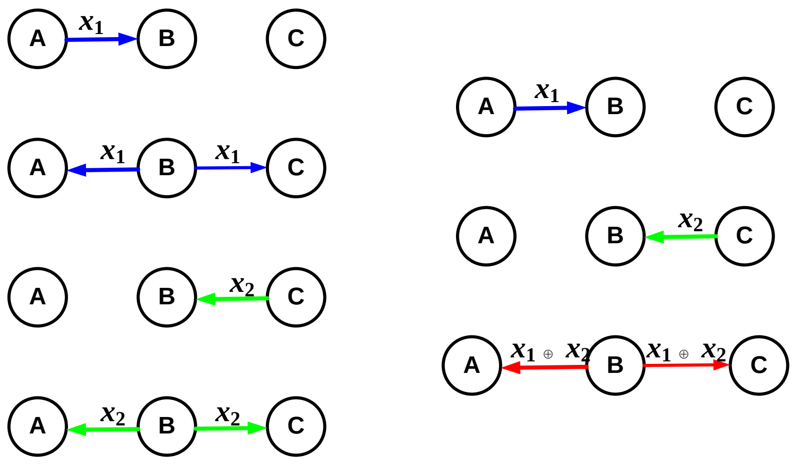
Figure 2-5: Single-edge wiretap butterfly network with secure network code.

2.3.3 WIRELESS RESOURCES

In a wireless environment, network coding can be used to offer benefits in terms of battery life of intermediate nodes or base stations, wireless bandwidth and delay. Consider the wireless ad-hoc network shown in Figure 2-6, where devices *A* and *C* need to exchange the binary files x_1 and x_2 via *B* as a relay. Presumably time is slotted and each device transmits and receives a file during a timeslot (half-duplex communication). As Figure 2-6(a) depicts, nodes *A* and *C* send their files to the relay *B*, and this forwards each file to the corresponding destination.

The network coding approach improves the natural capability of wireless channels for broadcasting and their resource utilization. As Figure 2-6(b) shows, node *C* receives both files x_1 and x_2, and performs on them a XOR binary operation to create the file $x_1 \oplus x_2$, which it then transmits to both receiver *A* and *C* using a common transmission. Node *A* has x_1 and can thus decode x_2. Node *C* performs as Node *A*.

Consequently, the network coding approach offers benefits in terms of energy efficiency (node *B* transmits once instead of twice), delay (the transmission involved three instead of four timeslots), and wireless bandwidth (the wireless channels are occupied for a smaller amount of time and the file $x_1 \oplus x_2$ is not consumed an excessive bandwidth to fulfil their transmissions).



(a) Without network coding (b) With network coding

Figure 2-6: Node *A* and *C* exchange information via relay *B*. The network coding approach saves one broadcast transmission.

2.3.4 ROBUSTNESS AND ADAPTABILITY

This is a vital topic to discuss under network coding benefits. Network coding can offer significant benefits in terms of operational complexity in dynamically changing environments, such as wireless networks, which frequently change because nodes move, turn on and off or roam out of range. In such environments, networks are restricted to use very simple distributed algorithms to avoid cost of storing because details of topology (availability of nodes and links) are changed rapidly.

Now Figure 2-6 is considered again with the assumption that node *A* and *C* may go into sleep mode (or may move out of range) at random without notifying the node *B* (wireless base station). If the base station *B* broadcasts x_1 or x_2, the transmission might be completely wasted, since the intended destination might not be able to receive it. However, if the base station broadcasts $x_1 \oplus x_2$ instead of x_1 or x_2, (or more generally, random linear combinations of the information packets) the transmission will bring new information to all active nodes. Either *A* or *C* will be woken up, so as to obtain x_1 or x_2 by decoding and send an acknowledgment to the base station. Then the base station can terminate their transmission to either *A* or *C*.

2.4 POTENTIAL DISADVANTAGES OF NETWORK CODING

This section is allocated to discuss the disadvantage of network coding. Network coding offers not only the benefits but also it comes with some potential disadvantages which it is vital to discuss in here.

2.4.1 COMPLEXITY

The two complexities, *set-up complexity* and *operational complexity*, accompany network coding [1]. The former is the complexity of designing the network coding scheme, which consists of selecting the paths through information flows, and determining the operations (coding, forwarding) that the nodes of the network perform. In a time-variant network[2], such as wireless ad-hoc networks, this complexity is higher because a routing table in each node should be updated in the time domain.

Operational complexity is defined as the amount of computational and network resources required per information unit successfully delivered. Again this complexity is strongly correlated with the network coding scheme employed. In linear network

[2] The nodes and links in the network are moved with reference to time

coding, a linear combination of h information streams at each coding node requires $O(h^2)$ operations over finite field $\mathbb{F}_{\mathbf{q}}$; to recover the source symbols, each receiver needs to solve a system of $h \times h$ linear equations, which requires $O(h^3)$ operations over $\mathbb{F}_{\mathbf{q}}$, if Gaussian elimination is used. Moreover the network nodes should be upgraded with additional functionalities (XOR, Gaussian elimination).

2.4.2 DELAY

Link delay is a general term used for any transmission network, but it causes fatal effects in network coding, and Section 1.2 discussed how it affects network coding.

In practical network coding, a delay-free assumption is denied and therefore coding and decoding delays are essentially considered. These delays are depended on a number of factors, which are: the network coding scheme, the finite field size, the number of coding nodes occupied to fulfil the multicast transmission, the average number of in-links per coding node, and the size of the decoding matrix (see section 2.1.3). Consequently, overall delays contribute to exhausting network and coding resources, and degrade network coding performances.

2.4.3 SECURITY

Unexpectedly network coding allows access to a fundamental network threat which is called a Byzantine modification, which can cause catastrophic fatal damage to network multicasting. This mixing of information can be catastrophic if the network consists of Byzantine nodes, i.e., malicious internal nodes that pretend to be routers but instead eavesdrop on transmissions and inject fake packets, with the objective of disrupting communications. In this case, even a small amount of corrupted information may be mixed with all the information flowing in the network, causing decoding errors.

2.5 NETWORK CODING APPLICATIONS

Network coding applications, their benefits and performance, are discussed thoroughly in [4] and [5]. Large content distribution systems, such as Bit Torrent and Microsoft Security Content Distribution (MSCD), are examples of *peer-to-peer* (P2P) systems. Minimum download times and more robustness are benefits that network coding offers to P2P systems. For bidirectional traffic in a wireless network (Figure 2-6(b)), network coding improves throughput when two wireless nodes communicate via a common base station. Other applications are residential wireless mesh networks, many-to-many broadcast, ad-hoc sensor networks, network tomography, and network security.

2.6 KEY PREVIOUS WORK ON NETWORK CODING

This section is allocated to discuss prior works in network coding relevant to this thesis. In their seminal research, Ahlswede *et al.* [1] illustrated that if network coding is permitted over the nodes of a network, the communication rate can be improved over that obtainable by routing alone. Li *et al.* [6] showed that linear coding is sufficient for multicast network coding problems, i.e., codes in which each packet sent over the network is a linear combination of the original packets. Koetter and Médard [7] introduced an algebraic framework for the study of network coding and gave a condition for valid codes. This framework was used by Ho *et al.* [8] to show that linear network codes can be efficiently constructed by employing a randomized algorithm. Jaggi *et al.* [9] proposed a deterministic polynomial-time algorithm for finding feasible network codes for multicast networks.

2.7 REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, “*Network information flow*”, *IEEE Trans. Information Theory,* vol. IT-46, no.4, pp.1204-1216, July 2000.

[2] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L.Tolhuizen, “Polynomial time algorithms for multicast network code construction,” *IEEE Trans. Information Theory*, vol. 51, no. 6, pp. 1973-1982, 2005.

[3] P. A. Chou, Y. Wu, and K. Jain. Practical network coding. In Allerton, Oct. 2003.

[4] C. Fragouli and E. Soljanin, “Network Coding Applications”, *Foundation and Trends in Networking, Vol 2,* no.2, pp.135-269,2007.

[5] C. Fragouli, J. Y. Le Boudec and J. Widmer, Network Coding: An Instant Primer, *ACM SIGCOMM Computer Communication Review*, Volume 36, Number 1, January 2006, pp. 63-68

[6] S.-Y. R. Li, N. Cai and R. W. Yeung, “Linear Network Coding”, *IEEE Trans. Information Theory,* vol. 49, no.4, pp.371-381, 2003.

[7] R. Koetter and M. Médard, “An algebraic approach to network coding”, *IEEE/ACM Trans. Networking,* vol. 11, no. 5, pp. 782–795, 2003.

[8] T. Ho, R. Koetter, M. Médard, D. Karger, and M. Effros, “The benefits of coding over routing in a randomized setting”, in *Proc. IEEE Int. Symp. Information Theory*, Yokohama, Japan, Jun./Jul. 2003, p. 442.

[9] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L.Tolhuizen, “Polynomial time algorithms for multicast network code construction,” *IEEE Trans. Information Theory*, vol. 51, no. 6, pp. 1973-1982, 2005.

[10] C. Fragouli and E. Soljanin, “Network Coding Fundamentals”, Foundations and Trends in Networking, vol. 2, no. 1, pp. 1-133, 2007.

3 ALGORITHMIC SOLUTIONS FOR NETWORK CODING PROBLEMS

Network coding problems are theoretically or practically solved by algorithmic solutions with computer networks being mapped into the algorithmically useful framework of graphs. These can be defined as a data structure and into which the problem to be solved is mapped. Graph based algorithms are developed based on graph theories [1] based on objects termed *vertices* or *nodes* which are related via *edges* or *links*. Computer networks are usually mapped into directed graphs, with nodes representing equipment such as routers and switches, and links representing wired or wireless network channels.

This chapter provides the first part of the underpinning for the solutions proposed in this thesis by discussing network code design algorithms for multicasting and the evolutionary approach for network code design. Section 3.1 discusses network multicast whilst section 3.2 covers network code design algorithms for multicasting. The subsequent chapter will introduce an evolutionary approach for network code design.

3.1 NETWORK MULTICAST

Network multicast refers to transmitting simultaneously the same information to multiple receivers in the network [2]. A simple example of multicasting is sending an e-mail message to a mailing list. In this research, the concept is expanded to cover transmitting simultaneously different sets of the same information to multiple receivers in the network. For example, an anti-virus software server needs to update simultaneously client security definitions. However, the clients use different platforms (e.g. Linux or Windows) with which the definition should be compatible. Therefore, it is necessary to transmit simultaneously compliant and varied definition sets in the same update version.

3.1.1 GRAPH REPRESENTATION FOR MULTICASTING

This section describes essential elements of the proposed solutions constituting the preliminary processes for all subsequent network coding solutions. A graph is considered as a directed acyclic graph (DAG) and there are no directed cycles or negative cycles [13]. The nodes of a DAG can be topologically sorted into a sequence $\langle v_1, v_2, ..., v_n \rangle$ such that $(v_i, v_j) \in E$ implies $i < j$. A topological order of a directed acyclic graph $G(V, E)$ can be computed in linear time $O(m+n)$ where $m = |E|, n = |V|$ using either depth-first search (DFS) or breadth-first search (BFS) algorithms. The nodes on any path in a DAG increase in topological order.

3.1.1.1 Adjacency Matrix Representation

An *n*-node graph can be represented by an $n \times n$ *adjacency matrix* M in which M_{ij} is 1 if $(i, j) \in E$ and 0 otherwise. Edge insertion or removal and edge queries work in constant time. It takes time $O(n)$ to obtain the edges entering or leaving a node. Each node in the graph has adjacent nodes whose edges can be represented as an adjacent vector. The graph is formed as the adjacent matrix combining all adjacent vectors. This representation can be generalized to store additional information such as edge weights in a separate matrix, the *weight matrix W*, and is an efficient and inexpensive representation method for dynamic and static graphs.

In Figure 3-1(a) shows a DAG, and its adjacent matrix representation and weight matrix are consecutively shown in Figure 3-2 (a) and Figure 3-2 (b). Each row of the adjacent matrix represents the adjacent vector belonging to each node in the DAG.

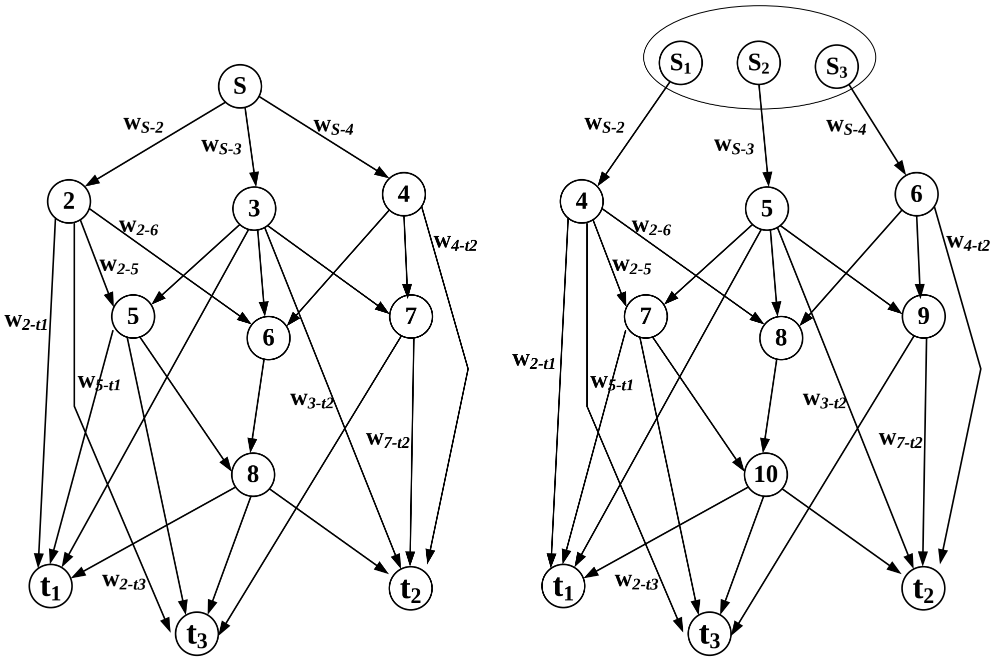


Figure 3-1: (a) Directed acyclic graph (DAG) with links weight; (b) The DAG for multicast network

$$
\begin{array}{c|ccccccccccc}
 & S & 2 & 3 & 4 & 5 & 6 & 7 & 8 & t_1 & t_2 & t_3 \\
\hline
S & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

(a)

$$
\begin{array}{c|ccccccccccc}
 & S & 2 & 3 & 4 & 5 & 6 & 7 & 8 & t_1 & t_2 & t_3 \\
\hline
S & 0 & w_{S,2} & w_{S,3} & w_{S,4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & w_{2,5} & w_{2,6} & 0 & 0 & w_{2,t_1} & 0 & w_{2,t_3} \\
3 & 0 & 0 & 0 & 0 & w_{3,5} & w_{3,6} & w_{3,7} & 0 & w_{3,t_1} & w_{3,t_2} & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & w_{4,6} & w_{4,7} & 0 & 0 & w_{4,t_2} & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{5,8} & w_{5,t_1} & 0 & w_{5,t_3} \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{6,8} & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{7,t_2} & w_{7,t_3} \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{8,t_1} & w_{8,t_2} & w_{8,t_3} \\
t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

(b)

Figure 3-2: (a) Adjacent matrix representation for DAG in Figure 3-1; (b) Weight matrix for DAG in Figure 3-1(a)

3.1.1.2 Adjacent Matrix Representation for Multicast Network - *M*

Figure 3-1(b) shows a multicast representation of the DAG in Figure 3-1(a). The source *S* is split into *h*-sub sources or individual data streams, and they are represented as individual nodes. In Figure 3-1(b) shows a 3-subsource multicast network, and its adjacent matrix and weight matrix are consecutively represented in Figure 3-3 (a) and (b). The proposed solutions for the network coding problems are implemented using MATLAB, and code details are given in Appendix B-2.

$$
\begin{array}{c|ccccccccccccc}
 & S_1 & S_2 & S_3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & t_1 & t_2 & t_3 \\
\hline
S_1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_3 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

(a)

$$
\begin{array}{c|ccccccccccccc}
 & S_1 & S_2 & S_3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & t_1 & t_2 & t_3 \\
\hline
S_1 & 0 & 0 & 0 & w_{S_1,4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_2 & 0 & 0 & 0 & 0 & w_{S_2,5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_3 & 0 & 0 & 0 & 0 & 0 & w_{S_3,6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & w_{4,7} & w_{4,8} & 0 & 0 & w_{4,t_1} & 0 & w_{4,t_3} \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & w_{5,7} & w_{5,8} & w_{5,9} & 0 & w_{5,t_1} & w_{5,t_2} & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{6,8} & w_{6,9} & 0 & 0 & w_{6,t_2} & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{7,10} & w_{7,t_1} & 0 & w_{7,t_3} \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{8,10} & 0 & 0 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{9,t_2} & w_{9,t_3} \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{10,t_1} & w_{10,t_2} & w_{10,t_3} \\
t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

(b)

Figure 3-3: (a) Adjacent matrix representation for the multicast DAG in Figure 3-1(b); (b) Weight matrix for the multicast DAG in Figure 3-1(b)

3.1.1.3 Adjacent Matrix Representation for Dynamic Multicast Networks

When nodes move randomly and their links with adjacent nodes thus appear and disappear, a dynamic multicast network is formed, meaning that each adjacent vector may vary in the time domain with consequent updating of the adjacent matrix required. Moreover every node re-computes a forwarding factor whenever the topology or link qualities change. The nodes that have forwarding factors smaller than a threshold are not included in the adjacent vectors, and the adjacent matrix is not updated by the source. In this complex environment, the adjacent matrix represents the network for a short time period during which each adjacent vector is assumed fixed.

As an example, node 10 in Figure 3-1(b) moves out of the range of node 7 and into the range of node 9. The adjacent vector for the former changes from $[0\ 0...10\ t_1\ 0\ t_3]$ to $[0\ 0...0\ t_1\ 0\ t_3]$ and for the latter from $[0\ 0...0\ t_2\ t_3]$ to $[0\ 0\ ......10\ t_2\ t_3]$. The source can modify the adjacent matrix from Figure 3-3 (a) to Figure 3-4 (and the weight matrix similarly) in a short time period (T). The time complexity for updating the adjacent matrix in the network *G*(*V*,*E*) can be calculated by $\mathrm{O}\left((h+|V|-1)\varsigma\right)$ where ς - is the number of nodes with updated adjacent vectors.

$$\begin{array}{c|cccccccccccccc}
 & S_1 & S_2 & S_3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & t_1 & t_2 & t_3 \\
\hline
S_1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_3 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \leftarrow 0 & 1 & 0 & 1 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \leftarrow 1 & 0 & 1 & 1 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Figure 3-4: Adjacent matrix representation for dynamic multicast network

3.1.2 IMPLEMENTATION OF ALGORITHMS WITH ADJACENT MATRIX

This section describes the implementation procedures of two vital algorithms, namely the augmenting path and linear disjoint path algorithms. The implementation process is based on a data structure and the efficiency of the implementation depends on a selected data structure. Since a matrix is a well organised data structure, facilitating data manipulation, the adjacent matrix data structure is employed. Moreover, the matrix is built in the source node, and programs can be easily implanted there or in a network interface card (NIC).

3.1.2.1 Implementation of an Augmenting Path Algorithm (APA)

This implementation is new but derives from the Breadth First Search (BFS) algorithm [13]. The algorithm operates on the adjacent matrix(*M*) to identify all available paths from each sub source $(\{S_i\}, 1 \le i \le h)$ to each receiver $(\{t_j\}, 1 \le j \le N)$. Row and column indices of M are assigned *node_IDs*. The algorithm sorts elements $m_{i,j} \in \{0,1\}$ of M along the rows. The algorithm looks for a ‘1’ entry along the rows, concatenating the node index with a preceding index (node_ID). If there is more than one ‘1s’ entry in the row, a preceding path is split into sub branches at the preceding index (node_ID). A set of individual paths is then formed at the preceding node_ID. If a newly identified index (node_ID) belongs to the receiver $(\{t_j\}, 1 \le j \le N)$ then a path is completed and stored. If a newly identified index (node_ID) is terminated at its own index (node_ID) then the path no longer exists and is erased. The algorithm is terminated when each newly identified index (node_ID) for all preceding indexes (node_IDs) belongs either to a receiver $(\{t_j\}, 1 \le j \le N)$ or its path is terminated at its own index. The algorithm’s time complexity can be calculated as $\mathrm{O}(h|V|^2)$.

$$\begin{array}{l|ccccccccccccc l}
 & S_1 & S_2 & S_3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & t_1 & t_2 & t_3 & \\
\hline
\underline{S_1} \Rightarrow & 0 & 0 & 0 & 1\Uparrow & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \Rightarrow \left\langle S_1, 4 \Downarrow \right\rangle \\
S_2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
S_3 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
\Rightarrow 4 & 0 & 0 & 0 & 0 & 0 & 0 & 1\Uparrow & 1\Uparrow & 0 & 0 & 1\Uparrow & 0 & 1\Uparrow & \Rightarrow \left\langle \begin{matrix} S_1, 4, 7 \Downarrow \\ S_1, 4, 8 \Downarrow \end{matrix} \right\rangle, \left\langle \begin{matrix} S_1, 4, t_1 \\ S_1, 4, t_3 \end{matrix} \right\rangle \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \\
\Rightarrow 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\Uparrow & 1\Uparrow & 0 & 1\Uparrow & \Rightarrow \left\langle S_1, 4, 7, 10 \Downarrow \right\rangle, \left\langle \begin{matrix} S_1, 4, 7, t_1 \\ S_1, 4, 7, t_3 \end{matrix} \right\rangle \\
\Rightarrow 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\Uparrow & 0 & 0 & 0 & \Rightarrow \left\langle S_1, 4, 8, 10 \Downarrow \right\rangle \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & \\
\Rightarrow 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1\Uparrow & \Rightarrow \left\langle \begin{matrix} S_1, 4, 7, 10, t_1 \\ S_1, 4, 7, 10, t_2 \\ S_1, 4, 7, 10, t_3 \end{matrix} \right\rangle, \left\langle \begin{matrix} S_1, 4, 8, 10, t_1 \\ S_1, 4, 8, 10, t_2 \\ S_1, 4, 8, 10, t_3 \end{matrix} \right\rangle \\
t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
t_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &
\end{array}$$

Figure 3-5: The augmenting path algorithm implementation with adjacent matrix in Figure 3-3 (a), the algorithm identifies all available paths between sub source *S₁* and receivers {*t₁*,*t₂*,*t₃*}

Figure 3-5 illustrates the operation of the algorithm on the adjacent matrix in Figure 3-3 (a). Initially, all available paths from the sub source $\{S_1\}$ to receivers $\{t_1, t_2, t_3\}$ are sorted. A '1' entry is found in the row *S_1* at node 4 and this index is concatenated to form a path $\left\langle S_1, 4 \right\rangle$. This process continues as shown where it should be noticed that in the node 4 row, two paths to receivers are found and stored, and also two further paths which continue to be augmented until receivers are reached.

3.1.2.2 A Linear Disjoint Paths Algorithm Implementation

The set of *h* paths between the source and receiver defined as a set of linear disjoint paths when none of the paths overlap.

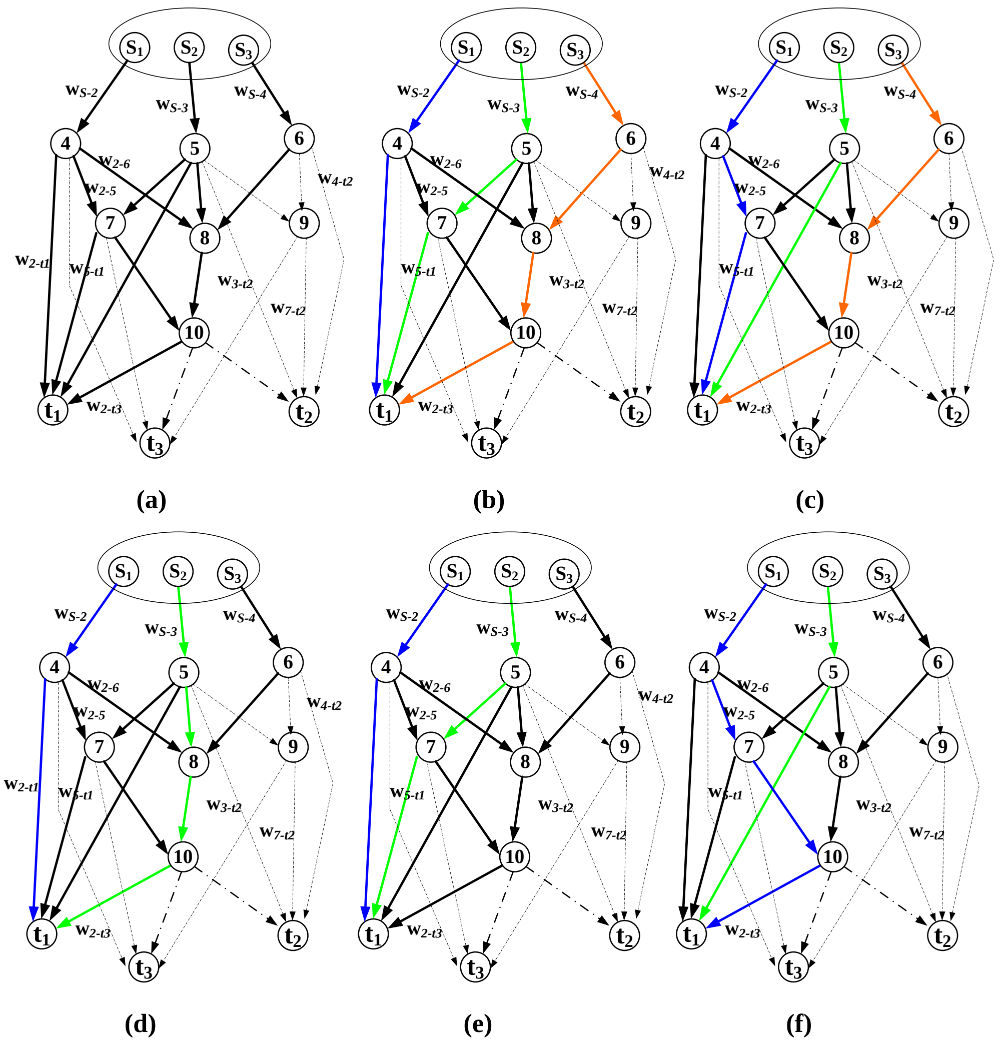


Figure 3-6: (a) All available paths between source and receiver – t_1; (b, c) Two different sets of 3 – linear disjoint paths; (d, e, f) Three different sets of 2 – linear disjoint paths

Figure 3-6 (a) shows all available paths between source and receiver t_1; Figure 3-6 (b) and Figure 3-6 (c) show two sets of three linear disjoint paths and Figure 3-6 (d)-(f) show three different sets of two linear disjoint paths. The source-receivers paths identified in Figure 3-5 are stored in a format shown in Figure 3-7; all sets of paths are shown in Figure 3-8.

	S_1	S_2	S_3	4	5	6	7	8	9	10	t_1	t_2	t_3
$\langle S_1,4,t_1\rangle$	1	0	0	1	0	0	0	0	0	0	1	0	0
$\langle S_1,4,7,t_1\rangle$	1	0	0	1	0	0	1	0	0	0	1	0	0
$\langle S_1,4,7,10,t_1\rangle$	1	0	0	1	0	0	1	0	0	1	1	0	0
$\langle S_1,4,8,10,t_1\rangle$	1	0	0	1	0	0	0	1	0	1	1	0	0
$\langle S_1,4,7,10,t_2\rangle$	1	0	0	1	0	0	1	0	0	1	0	1	0
$\langle S_1,4,8,10,t_2\rangle$	1	0	0	1	0	0	0	1	0	1	0	1	0
$\langle S_1,4,t_3\rangle$	1	0	0	1	0	0	0	0	0	0	0	0	1
$\langle S_1,4,7,t_3\rangle$	1	0	0	1	0	0	1	0	0	0	0	0	1
$\langle S_1,4,7,10,t_3\rangle$	1	0	0	1	0	0	1	0	0	1	0	0	1
$\langle S_1,4,8,10,t_3\rangle$	1	0	0	1	0	0	0	1	0	1	0	0	1

Figure 3-7: All available paths from source –S_1 to receivers {t_1, t_2, t_3}

$\langle S_1,4,t_1\rangle$	$\langle S_2,5,7,t_1\rangle$	$\langle S_3,6,8,10,t_1\rangle$	$\cdots\cdots\langle S_h,...t_1\rangle$
$\langle S_1,4,7,t_1\rangle$	$\langle S_2,5,t_1\rangle$		$\vdots$
$\langle S_1,4,7,10,t_1\rangle$	$\langle S_2,5,8,10,t_1\rangle$		$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\cdots\cdots\cdots\vdots$
$\langle S_1,4,7,10,t_2\rangle$	$\langle S_2,5,8,10,t_2\rangle$	$\langle S_3,6,9,t_2\rangle$	$\cdots\cdots\langle S_h,...t_2\rangle$
$\langle S_1,4,8,10,t_2\rangle$	$\langle S_2,5,t_2\rangle$	$\langle S_3,6,8,10,t_2\rangle$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\langle S_1,4,t_3\rangle$	$\langle S_2,5,8,10,t_3\rangle$	$\langle S_3,6,8,10,t_3\rangle$	$\cdots\cdots\langle S_h,...t_3\rangle$
$\langle S_1,4,7,t_3\rangle$	$\langle S_2,5,7,t_3\rangle$	$\langle S_3,6,9,t_3\rangle$	
$\langle S_1,4,7,10,t_3\rangle$	$\langle S_2,5,7,10,t_3\rangle$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\langle S_1,....t_N\rangle$	$\langle S_2,....t_N\rangle$	$\langle S_3,....t_N\rangle$	$\cdots\cdots\langle S_h,...t_N\rangle$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figure 3-8: Available paths between sources {S_1……S_h} to receivers {t_1……t_N}

The linear disjoint path algorithm hierarchically examines each path in Figure 3-8 to form the sets of linear disjoint paths in Figure 3-10. The algorithm picks the first path $\langle S_1, 4, t_1 \rangle$ with its related vector from Figure 3-8 and Figure 3-7, and put them into a table Figure 3-9. Each path is selected in a hierarchical way as shown in Figure 3-9. Two paths are compared in a vector format and its computational complexity $\mathrm{O}(2n)$. The vector 'Test(m)' is formed by multiplying individual elements in same indexes. If Test(m) is formed as a unit vector and entry '1' is in receiver j, then these two paths are the set of 2-linear disjoint path for that receiver else the paths tested are not linearly disjoint. For example, Test (1) in Figure 3-9 is the unit vector and entry '1' is in receiver t_1, so paths $\langle S_1, 4, t_1 \rangle$ and $\langle S_2, 5, 7, t_1 \rangle$ are linearly disjoint. Similarly, Test (2) to Test (6) indicate sets of linearly disjoint paths but Test (7) shows a set of paths $\langle S_2, 5, 8, 10, t_1 \rangle, \langle S_3, 6, 8, 10, t_1 \rangle$ that is not linearly disjoint. Consequently the linear disjoint path algorithm hierarchically examines (Figure 3-9) each path in Figure 3-8 to form the table in Figure 3-10.

	S_1	S_2	S_3	4	5	6	7	8	9	10	t_1	t_2	t_3	
$\Rightarrow \langle S_1,4,t_1 \rangle$	1	0	0	1	0	0	0	0	0	0	1	0	0	
$\langle S_2,5,7,t_1 \rangle$	0	1	0	0	1	0	1	0	0	0	1	0	0	
$Test(1) \Downarrow$	0	0	0	0	0	0	0	0	0	0	1	0	0	$\Rightarrow \langle S_1,4,t_1 \rangle \langle S_2,5,7,t_1 \rangle$
$\langle S_2,5,t_1 \rangle$	0	1	0	0	1	0	0	0	0	0	1	0	0	
$Test(2)$	0	0	0	0	0	0	0	0	0	0	1	0	0	$\Rightarrow \langle S_1,4,t_1 \rangle \left\{ \begin{array}{l} \langle S_2,5,7,t_1 \rangle \\ \langle S_2,5,t_1 \rangle \end{array} \right\}$
$\langle S_2,5,8,10,t_1 \rangle$	0	1	0	0	1	0	0	1	0	1	1	0	0	
$Test(3)$	0	0	0	0	0	0	0	0	0	0	1	0	0	$\Rightarrow \langle S_1,4,t_1 \rangle \left\{ \begin{array}{l} \langle S_2,5,7,t_1 \rangle \\ \langle S_2,5,t_1 \rangle \\ \langle S_2,5,8,10,t_1 \rangle \end{array} \right\}$
$\langle S_3,6,8,10,t_1 \rangle$	0	0	1	0	0	1	0	1	0	1	1	0	0	
$Test(4)$	0	0	0	0	0	0	0	0	0	0	1	0	0	$\Rightarrow \langle S_1,4,t_1 \rangle \left\{ \begin{array}{l} \langle S_2,5,7,t_1 \rangle \\ \langle S_2,5,t_1 \rangle \\ \langle S_2,5,8,10,t_1 \rangle \end{array} \right\Vert \langle S_3,6,8,10,t_1 \rangle \Big\}$
$\Rightarrow \langle S_2,5,7,t_1 \rangle$	0	1	0	0	1	0	1	1	0	0	1	0	0	
$\langle S_3,6,8,10,t_1 \rangle$	0	0	1	0	0	1	0	1	0	1	1	0	0	
$Test(5)$	0	0	0	0	0	0	0	0	0	0	1	0	0	$\Rightarrow \langle S_1,4,t_1 \rangle \left\{ \begin{array}{l} \langle S_2,5,7,t_1 \rangle \langle S_3,6,8,10,t_1 \rangle \\ \langle S_2,5,t_1 \rangle \\ \langle S_2,5,8,10,t_1 \rangle \Vert \langle S_3,6,8,10,t_1 \rangle \end{array} \right\}$
$\Rightarrow \langle S_2,5,t_1 \rangle$	0	1	0	0	1	0	0	0	0	0	1	0	0	
$\langle S_3,6,8,10,t_1 \rangle$	0	0	1	0	0	1	0	1	0	1	1	0	0	
$Test(6)$	0	0	0	0	0	0	0	0	0	0	1	0	0	$\Rightarrow \langle S_1,4,t_1 \rangle \left\{ \begin{array}{l} \left\{ \begin{array}{l} \langle S_2,5,7,t_1 \rangle \\ \langle S_2,5,t_1 \rangle \end{array} \right\} \langle S_3,6,8,10,t_1 \rangle \\ \langle S_2,5,8,10,t_1 \rangle \Vert \langle S_3,6,8,10,t_1 \rangle \end{array} \right\}$
$\langle S_2,5,8,10,t_1 \rangle$	0	1	0	0	1	0	0	$1\downarrow$	0	1	1	0	0	
$\langle S_3,6,8,10,t_1 \rangle$	0	0	1	0	0	1	0	$1\downarrow$	0	1	1	0	0	
$Test(7)$	0	0	0	0	0	0	0	$1\Uparrow$	0	0	1	0	0	$\Rightarrow \langle S_1,4,t_1 \rangle \left\{ \begin{array}{l} \left\{ \begin{array}{l} \langle S_2,5,7,t_1 \rangle \\ \langle S_2,5,t_1 \rangle \end{array} \right\} \langle S_3,6,8,10,t_1 \rangle \\ \langle S_2,5,8,10,t_1 \rangle \end{array} \right\}$

Figure 3-9: The linear disjoint path algorithm hierarchically examines each path in Figure 3-8

$$\boxed{\begin{array}{l}
\Rightarrow t_1 \\
\langle S_1,4,t_1\rangle \left\{\begin{matrix} \left\{\begin{matrix}\langle S_2,5,7,t_1\rangle \\ \langle S_2,5,t_1\rangle\end{matrix}\right\}\left\{\langle S_3,6,8,10,t_1\rangle\right\} \\ \langle S_2,5,8,10,t_1\rangle \end{matrix}\right\} \\
\langle S_1,4,7,t_1\rangle \left\{\begin{matrix} \left\{\langle S_2,5,t_1\rangle\right\}\left\{\langle S_3,6,8,10,t_1\rangle\right\} \\ \langle S_2,5,8,10,t_1\rangle \end{matrix}\right\} \\
\langle S_1,4,7,10,t_1\rangle\langle S_2,5,t_1\rangle
\end{array}}$$

$$\vdots \qquad\qquad \vdots$$

$$\boxed{\begin{array}{l}
\Rightarrow t_2 \\
\left\{\begin{matrix}\langle S_1,4,7,10,t_2\rangle \\ \langle S_1,4,8,10,t_2\rangle\end{matrix}\right\}\left\{\left\{\langle S_2,5,t_2\rangle\right\}\left\{\langle S_3,6,9,t_2\rangle\right\}\right\} \\
\qquad\qquad\qquad \langle S_2,5,8,10,t_2\rangle\langle S_3,6,9,t_2\rangle
\end{array}}$$

$$\vdots \qquad\qquad \vdots$$

$$\boxed{\begin{array}{l}
\Rightarrow t_3 \\
\langle S_1,4,t_3\rangle \left\{\begin{matrix} \left\{\begin{matrix}\langle S_2,5,8,10,t_3\rangle \\ \langle S_2,5,7,t_3\rangle \\ \langle S_2,5,7,10,t_3\rangle\end{matrix}\right\}\langle S_3,6,9,t_3\rangle \\ \langle S_2,5,7,t_3\rangle\langle S_3,6,8,10,t_3\rangle \end{matrix}\right\} \\
\langle S_1,4,7,t_3\rangle\langle S_2,5,8,10,t_3\rangle\langle S_3,6,9,t_3\rangle \\
\langle S_1,4,7,10,t_3\rangle\langle S_3,6,9,t_3\rangle
\end{array}}$$

Figure 3-10: Sets of liner disjoint paths from sources {S_1……S_h} to each receiver

3.1.3 THE MIN-CUT MAX-FLOW THEOREM

Let $G = (V, E)$ be a graph (network) with set of vertices *V* and the set of edges $E \subset V \times V$ and assume that each edge has unit capacity. Consider a node $S \in V$ that wants to transmit information to a node $R \in V$. Theorem 3.1.3 was proved in 1972 by Menger [3] and in 1956 by Ford and Fulkerson [4].

Theorem 3.1.3: Consider a graph $G = (V, E)$ with unit capacity edges, a source vertex *S*, and a receiver vertex *R.* If a *min-cut* between *S* and *R* equals *h,* then the information can be sent from *S* to *R* at a maximum rate of *h.* Equivalently, there exist exactly *h- linear disjoint paths* between *S* and *R.*

Definition 3.1.3: A *cut* between *S* and *R* is a set of graph edges whose removal disconnects *S* from *R.* A *min-cut* is a cut with smallest (minimal) value. The *value* of the cut is the sum of the capacities of the edges in the cut.

If sink t_1 and t_2 are considered as an individual basis in Figure 2-1(a) or (b), each sink has two linear disjoint paths and the min-cut between *S* and t_1 or *S* and t_2 equals 2. Then the information can be sent from *S* to t_1 or *S* and t_2 at a maximum rate of 2. However, if source *S* needs to multicast its data to both t_1 and t_2 , each sink has a different min-cut value. When t_2 has min-cut two, then t_1 has min-cut one. If the sinks have different min-cuts, then source can multicast at the rate equal to the minimum of the min-cut but cannot always transmit to each receiver at the rate equal to its min-cut. But Figure 2-1(c) shows network coding allows each receiver t_1 and t_2 to maintain the min-cut two. It should be noted that the condition that the min-cuts equal the maximum flow is not always satisfied.

3.1.4 MAIN NETWORK CODING THEOREM IN THE MULTICAST SCENARIO

The main theorem in network coding was proved by Ahlswede *et.al.*[5] and Li *et.al.*[6].

Theorem 3.1.4: A communications network is represented by a directed acyclic graph $G = (V, E)$ with unit capacity edges and the value of the min-cut between the source node and each of the receivers is *h*. A set of *h* unit rate information sources $\{S_1, S_2, \dots, S_h\}$ is located on the same network node *S* (source) and simultaneously transmits information to a set of *N* receivers $\{t_1, t_2, \dots, t_N\}$. Then there exists a multicast transmission scheme over a large enough finite field $\mathbb{F}_q$, in which intermediate network nodes linearly combine their incoming information symbols over $\mathbb{F}_q$, that delivers the information from the sources simultaneously to each receiver at rate equal to *h*.

The min-cut max-flow theorem states that each of *N* receivers $\{t_1, t_2, \dots, t_N\}$ has at least *h*-linear disjoint paths and during the simultaneous multicasting from source to *N* receivers $\{t_1, t_2, \dots, t_N\}$, edges or nodes or both edges and nodes may be overlapped. These nodes and links are defined as coding nodes and out-link of coding nodes in order. This fundamental concept is used to develop the solutions proposed in this thesis.

3.1.5 AN EQUIVALENT ALGEBRAIC STATEMENT OF THE THEOREM

To route the *h* information sources to a particular receiver, a source has to identify *h*- linear disjoint paths between the source and the receiver. The source can perform the *augmenting path algorithm* and *linear disjoint path algorithm* for identifying these disjoint paths. Figure 3-11 (a), (b) and (c) show a set of 3-linear disjoint paths for receivers t_2, t_1 and t_3 in order. Assume sources $\left\{S_1^a, S_2^b, S_3^c\right\}$ need to transmit unit packets $\left\{a,b,c\right\}$ simultaneously to the receivers $\{t_1,t_2,t_3\}$ and the packets can be routed through only selected disjoint paths in Figure 3-11 (d). In linear network coding, shared or overlapped links such as *EG, Gt₁, Gt₂, Gt₃* and *Ft₃* can transmit a linear combination of their input packets $\left\{a,b,c\right\}$ over $\mathbb{F}_q$. Such operations may be performed several times through the network if paths bring different information symbols.

The coefficients used to form this linear combination constitute what is called a local coding vector $c^l(e)$ for edge e. The dimension of $c^l(e)$ is $1\times\left|In(e)\right|$, where $In(e)$ is the set of incoming edges to the parent node of e. The vector of coefficients over $\mathbb{F}_q$, which they multiply incoming symbols to the parent node of e and form the linearly combined symbol. In Figure 3-11 (d),The local coding vector coefficients associated with edges {*EG*}, {*Gt₁, Gt₂, Gt₃*} and {*Ft₃, Ft₂*} are $c^l(EG)=[\alpha_1\ \ \alpha_2]$, $c^l(\left\{Gt_1,Gt_2,Gt_3\right\})=[\alpha_3\ \ \alpha_4]$ and $c^l(Ft_3,Ft_2)=[\alpha_5\ \ \alpha_6]$.

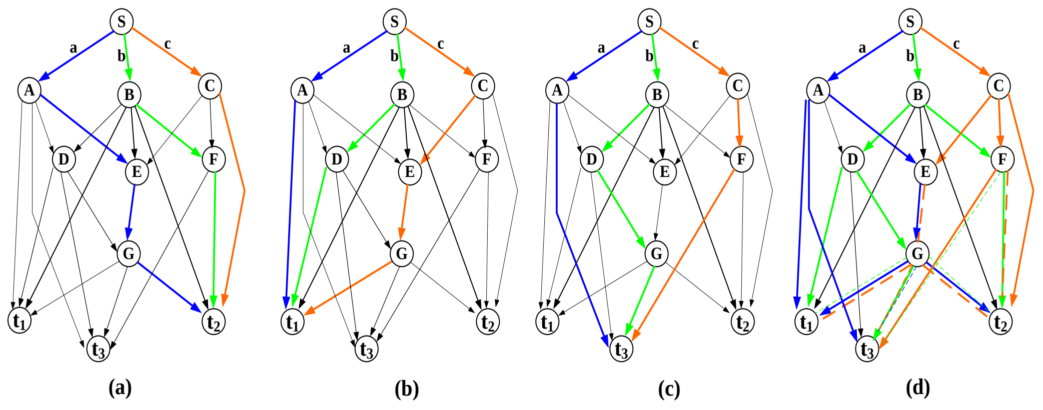


Figure 3-11: The possible linear disjoint paths for each receiver and path overlap over edge EG; (d) The linear network coding solution sends over edges EG, Gt₁, Gt₂, Gt₃, Ft₂ and Ft₃ .

The multicast transmission initiates from the source symbols and they are coded at the intermediate nodes in the network *G*. Generally the symbol flowing through any edges *e* of *G*, given by

$$\beta_1(e)x_1 + \beta_2(e)x_2 + \dots\dots\dots + \beta_h(e)x_h = \underbrace{[\beta_1(e)\,\beta_2(e)\dots\dots\dots\beta_h(e)]}_{\beta(e)}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_h \end{bmatrix}$$

Where $\beta(e) = [\beta_1(e)\,\beta_2(e)\dots\dots\dots\beta_h(e)]$ is *h*- dimensional vector over $\mathbb{F}_q$ and it is referred to as the *global coding vector* of edge *e*, or simplicity as the *coding vector*. In Figure 3-11 (d), the global coding vectors associated with edges {*EG*}, {*Gt₁*, *Gt₂*, *Gt₃*} and {*Ft₃,Ft₂*}are $\beta(EG) = [\alpha_1 \;\; 0 \;\; \alpha_2]$, $\beta(\{Gt_1, Gt_2, Gt_3\}) = [\alpha_1\alpha_4 \;\; \alpha_3 \;\; \alpha_2\alpha_4]$ and $\beta(Ft_3, Ft_2) = [0, \alpha_5, \alpha_6]$.

The global coding vectors $[\beta(e_1)\,\beta(e_2)\dots\dots\dots\beta(e_h)]$ associated with the input edges of each receiver node and their input symbols $\{\rho(e_1), \rho(e_2)\dots\dots\rho(e_h)\}$ related to the global coding vectors define a system of linear equations which can be used by each receiver to determine the original source symbols. The system of linear equations for the receiver t_j is shown in Figure 3-12. The receiver t_j solves the system of linear equations to obtain the original source symbols $x_i, 1 \le i \le h$. Therefore, all $A_j, 1 \le j \le N$ are full rank (Appendix A-1), allowing all receivers $t_j, 1 \le j \le N$ to obtain the original source symbols $x_i, 1 \le i \le h$.

$$\begin{bmatrix} \rho_1^j \\ \rho_2^j \\ \vdots \\ \rho_h^j \end{bmatrix} = \begin{bmatrix} \beta^j(e_1) \\ \beta^j(e_2) \\ \vdots \\ \beta^j(e_h) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_h \end{bmatrix} = A_j \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_h \end{bmatrix}$$

Figure 3-12 : The system of linear equations for the receiver t_j.

Consequently, matrices $A_j, 1 \le j \le N$ can be expressed in terms of the components of the local coding vector coefficient $\{\alpha_k\}$. In Figure 3-11(d), the three receivers $\{t_1, t_2, t_3\}$ observe the matrixes $\{A_1, A_2, A_3\}$ which show in Figure 3-13. All matrixes $A_j, 1 \le j \le N$ satisfy the condition of $\det[A_j(\alpha_k)] \ne 0; \quad 1 \le j \le N$, and then they are full rank matrixes. In Figure 3-13 matrixes $\{A_1, A_2, A_3\}$ satisfy the condition.

$$A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \alpha_1\alpha_4 & \alpha_3 & \alpha_2\alpha_4 \end{bmatrix} \quad A_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \alpha_5 & \alpha_6 \\ \alpha_1\alpha_4 & \alpha_3 & \alpha_2\alpha_4 \end{bmatrix} \quad A_3 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & \alpha_5 & \alpha_6 \\ \alpha_1\alpha_4 & \alpha_3 & \alpha_2\alpha_4 \end{bmatrix}$$

Figure 3-13: The coding matrixes for receivers - t_1, t_2 and t_3.

Then, the main multicast theorem can be expressed in algebraic language as, in linear network coding, the components $\{\alpha_k\}$ of the local coding vectors are in some large enough finite field $\mathbb{F}_q$, the global coding vectors consist of the components $\{\alpha_k\}$ of the local coding vectors over field $\mathbb{F}_q$, all matrixes $A_j, 1 \le j \le N$ consist of *h*-

global coding vectors, then all receivers obtain simultaneously the original source symbols $x_i, 1 \le i \le h$ if all matrixes $A_j, 1 \le j \le N$ are full rank.

3.2 NETWORK CODE DESIGN ALGORITHMS FOR MULTICASTING

Network code design algorithms are based on the main network coding theorems in section 3.1.4 and the assumptions of the network multicast model discussed in section 3.1. These algorithms are classified as *centralised* and *decentralised*, based on the information they require to execute. The former operate on the global information of the entire network structure (topological information), whereas the latter operate only on local information (without topological information). Here, examples are presented of relevant centralised and decentralised algorithms, and their efficiency is discussed.

3.2.1 MEASURING THE EFFICIENCY OF THE ALGORITHMS

An algorithm can be defined as a sequential procedure or a specific set of instructions for solving a problem. Network code design problems are solved by algorithms such as Linear Information Flow (LIF) and Random Assignment (Random Linear Network Coding), and they are categorised as centralised and decentralised algorithms [2].

The efficiency of the algorithms, and their complexity, are vital topics. Complexity analysis addresses how much time is required by the algorithm to solve a problem and is based on counting primitive operations (arithmetic, logical, reads, writes, etc.). Example: The complexity of an augmenting path algorithm is expressed

$\mathrm{O}(|E|)$, where $|E|$ is a number of edges in the graph $G(V,E)$; the complexity of LIF algorithm is expressed $\mathrm{O}(|E|Nh^2)$.

3.2.2 CENTRALISED ALGORITHM

3.2.2.1 Linear Information Flow Algorithm

LIF is a greedy algorithm [Appendix A-4.2] that observes the choice of coding vectors which should be able to preserve the multicast property of the network. The algorithm sequentially visits the coding points in a topological order[3] and assigns coding vectors to them. All visited coding points are only assigned the coding vectors and it is granted that, these coding vectors preserve the multicast properties (such as min-cut condition) for all downstream receivers. Intuitively, the algorithm preserves *h* "degree of freedom" on the paths from the sources $\{S_1, S_2, \ldots, S_h\}$ to each receiver.

The explanation of LIF begins by the initial procedures in *Algorithm* 3.2.2.1. Assuming a given multicast instance $\{G=(V,E),S,t\}$, the first common steps are:

(1) Find *h* edge- disjoint paths $\{(S_i,t_j), 1 \le i \le h, 1 \le j \le N\}$ from the sources to receivers, the associated sub graph $G'=(V',E') \leftarrow \bigcup_{\substack{1\le i\le h \\ 1\le j\le N}} (S_i,t_j)$, the set of all coding points (Υ), and

(2) Find the associated minimal configuration.

[3] A topological order is simply a partial order in any acyclic graph *G* and such an order exists for the edges of *G*

Step (1) can be implemented using max flow algorithm which is given by the min-cut max-flow theorem (Theorem 3.1.3). A flow – augmenting path can be found in time $\mathrm{O}\left(|E|\right)$, therefore the total complexity is $\mathrm{O}\left(|E|hN\right)$.

Step (2) is essential to significantly reduce the required network resources, such as the number of coding points and the number of employed edges by the information flow. Algorithm 3.2.2.1 describes a brute force implementation, which sequentially attempts to remove each edge and examine the min-cut condition to each receiver is still satisfied. This implementation requires $\mathrm{O}\left(|E|^2 hN\right)$ operations.

```
$Algorithm$ 3.2.2.1: $Initial\ Processing\ \left(G,\ S,\ t\right)$
$Find\ (S_i, t_j)\ for\ all\ \ i, j$
$G' = (V', E') \leftarrow \bigcup_{\substack{1\le i\le h\\ 1\le j\le N}} (S_i, t_j)$
$\forall e \in E' \begin{cases} if\ (V', E' \setminus \{e\}\ satisfies\ the\ multicast\ property \\ then \begin{cases} E' \leftarrow E' \setminus \{e\} \\ G' \leftarrow (V', E') \end{cases} \end{cases}$
$return\ (G')$
```

Some additional notations are mentioned here to precisely describe the LIF in *Algorithm* 3.2.2.2. Let Υ and $t(\delta)$ denote the set of all coding points and the set of all receivers that employ a coding point δ in one of their paths. Each coding point δ appears in at most one path $\{(S_i, t_j), 1 \le i \le h, 1 \le j \le N\}$ for each receiver t_j. Let $f_{\leftarrow}^{j}(\delta)$ denote the predecessor coding point to δ along this path $\{(S_i, t_j), 1 \le i \le h, 1 \le j \le N\}$. The algorithm maintains a set C_j of *h*-coding points and a set $B_j = \{c_1^j, ...c_h^j\}$ of h coding vectors for each receiver t_j. The set C_j keeps tracks of the most recently visited coding point in each of the *h* edge disjoint paths $\{(S_i, t_j), 1 \le i \le h, 1 \le j \le N\}$, and the set B_j keeps the associated coding vectors.

Initially, the set C_j consists of the source nodes $\{S_1, S_2, \dots, S_h\}$ and the set B_j consists of vectors c_i^j relevant to the source nodes (orthonormal basis $\{e_1, e_2 ...., e_h\}$). They are unit vectors and represent c_i^j as $[1^{i-1} \; 0^i \qquad 0^{h-i}]$ where the $(i-1)^{th}$ position of the vector is represented by the source node $(\{S_i\}, 1 \le i \le h)$, and it is set 1 and rests of them are set 0s. For example the vector c_1^j is $[1\ 0\ 0\ ....0^{h-1}]$, and it is represented the source $\{S_1\}$. Moreover the set B_j is formed based on the *h*- dimensional space $\mathbb{F}_q^h$ (finite field $\mathbb{F}_q^h$) for preserving the multicast properties, and the algorithm maintains the condition at its all steps for all receivers t_j. The algorithm visits the coding points $\delta_k \in \Upsilon$ at step *k*, and assigns a coding vector $c(\delta_k)$ to the coding point while replacing the precedence vector, for all receivers $t_j \in t(\delta_k)$. The condition $\left|\mathbb{F}_q^h\right| > |N|$ is satisfied then the vector $c(\delta_k)$ always exists [2]. The set B_j contains the set of linear equations and the receiver t_j needs to solve the equations to obtain the original source symbols.

```
$Algorithm\ 3.2.2.2: LIF\ \left(\{G,\ S,\ t\}, \mathbb{F}_q\ with\ q > N\right)$
$\Upsilon \leftarrow coding\ points\ of\ \{G,\ S,\ t\}\ in\ topological\ order$
$\forall \delta \in \Upsilon,\ t(\delta) \leftarrow \{t_j\ such\ that\ \delta\ is\ in\ a\ path(S_i, t_j)\}$
$\forall t_j \in t$
    $B_j \leftarrow \{e_1, e_2 .......e_h\},$
    $C_j \leftarrow \{S_1^e, S_2^e, .....S_h^e\}$
    $\forall \delta \in C_j,\ a_j(\delta) = e_j$
$\bullet\ Repeat\ for\ K\ steps, 1 \le K \le |\Upsilon|$
$At\ step\ K\ access\ \delta_K \in \Upsilon$
    $Find\ c(\delta_K)\ such\ that\ c(\delta_K) \cdot a_j(\delta_K) \neq 0\ \forall t_j \in t(\delta_K)$
$and$
    $\forall t_j \in t(\delta_K)$
        $C_j \leftarrow (C_j \setminus \{f_{\leftarrow}^j(\delta_K)\} \cup \delta_K$
        $B_j \leftarrow (B_j \setminus \{c(f_{\leftarrow}^j(\delta_K))\} \cup c(\delta_K)$
        $\alpha_j(\delta_K) \leftarrow \frac{1}{c(\delta_K) \cdot a_j(\{f_{\leftarrow}^j(\delta_K)\})} a_j(\{f_{\leftarrow}^j(\delta_K)\})$
        $\forall \delta \in C_j \setminus \delta_K :$
        $\alpha_j(\delta) \leftarrow \alpha_j(\delta) - (c(\delta_K) \bullet \alpha_j(\delta))\alpha_j(\delta_K)$
$return(the\ coding\ vectors\ c(\delta), \forall \delta \in \Upsilon)$
```

3.2.3 DECENTRALISED ALGORITHM

3.2.3.1 Random Assignment

This algorithm is scalable, yields a very simple implementation and is well matched to practical applications such as dynamically changing networks. Each coding point randomly selects the coding vector coefficients (α_i) as its local coding vector. The algorithm operates over the finite field $\mathbb{F}_q$ and the field size $|q|$ is large enough for even choices of the coding vector coefficients to offer the linearly independent coding vectors, and consequently all receivers would be able to generate full rank coding matrixes. Theorem 3.2.3 states that the associated probability of decoding error can be made arbitrarily small by selecting an adequate large alphabet size [2].

Theorem 3.2.3: Consider an instance $\{G=(V,E),S,t\}$ with $N=|t|$ receivers, where the components of local coding vectors are chosen uniformly at random from the finite field $\mathbb{F}_q$ with $q>N$. The probability that all N receivers can decode all h sources is at least $(1-N/q)^{\eta'}$, where $\eta', (\eta' \le |E|)$ is the maximum number of coding points employed by any receiver.

The random coefficients $[(\alpha_1,.......\alpha_\mu), \eta' < \mu]$ are chosen by the algorithm and each network code is valid if the following condition is satisfied by the code. A_j is the decoding matrix for receiver $j, 1 \le j \le N$.

$$f(\alpha_1,.....\alpha_\mu) = \det A_1 \det A_2 ......\det A_N \ne 0$$

3.2.4 DECENTRALISED DETERMINISTIC ALGORITHM

This section discusses a subtree decomposition method for the network code design. The basic idea of this method is partitioning the network graph into subgraphs through which the same information flows. The structure of the network inside these subgraphs is not concerned with the network code design, and the network code design method addresses the connection of subgraphs and which receivers are in each subgraph. To illustrate this idea, the familiar example of a network with two sources and two receivers is used in Figure 3-14 (a), because there are three different information flows in the network. The first flow carries uncoded symbols from the source S_1, the second flow carries uncoded symbols from the source S_2 and the third flow carriers a linear combination of the symbols from the source S_1 and S_2. The first two flows are referred as the *source flows*, and the third flow is referred as the *coding flow*. Also each subgraph is a tree, rooted at the coding point or the source, and terminating either at receiver or other coding point. A subtree T_i is called a *source subtree* if it starts with the source and a *coding subtree* if it starts with a coding point. Figure 3-14 (b) shows how these flows are connected and how the receivers access to the flows. In Figure 3-14 (b), T_1 and T_2 are the source subtrees and T_3 is the coding subtree. This whole process is defined as the subtree decomposition or *information flow decomposition* method.

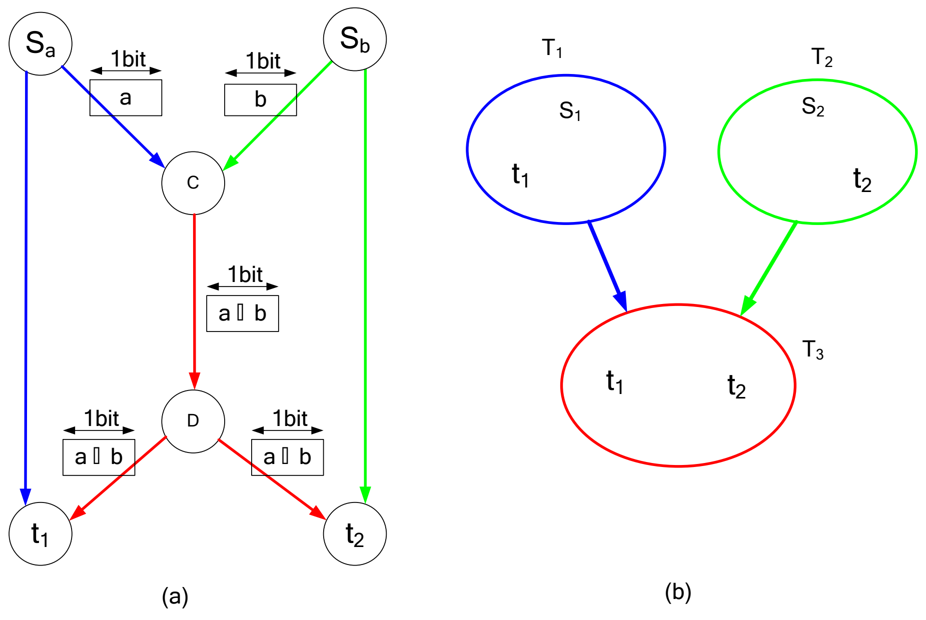


Figure 3-14: (a) A network with two sources and two receivers; (b) An information flow decomposition diagram for the network in Fig 3-14 (a)

The network code design assigns an h - dimensional coding vector $c(T_i) = [c_1(T_i).........c_h(T_i)]$ to each subtree T_i. The receiver $t_j, 1 \le j \le N$ observes h – coding vectors from h distinct subtrees to form the rows of the matrix $A_j, 1 \le j \le N$. A valid code for the network in Figure 3-14 (a) can be obtained by assigning the following coding vectors to the subtree in Figure 3-14 (b):

$$c(T_1) = [1\ \ 0], \quad c(T_2) = [0\ \ 1], \quad \textit{and}\ c(T_3) = [1\ \ 1].$$

The matrix for the receivers t_1 and t_2 are:

$$A_1 = \begin{bmatrix} c(T_1) \\ c(T_3) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \qquad A_2 = \begin{bmatrix} c(T_2) \\ c(T_3) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

Most of the network code design algorithm performances degrade with graph size, but the information flow decomposition method reduces the dimensionality of the network code design problem.

3.3 MULTICAST NETWORK CODE CONSTRUCTION IN THE LITERATURE

The first polynomial time algorithm [Appendix A-4.1], namely LIF for network code design was proposed by Sanders *et al.* [7], and independently by Jaggi *et al.* [8][9]. These algorithms were later extended by Barbero and Ytrehus in [10] attempting to minimise the required field size. Randomised algorithms were proposed for network code design by Ho *et al.* [11], and also by Sanders *et al.* [7]. Decentralized deterministic code design was introduced by Fragouli and Soljanin [12], who also first introduced minimal configurations and the brute force algorithm to identify the network codes.

3.4 REFERENCES

[1] R Diestel, “Graph Theory”, *Springer-Verlag,Heidelberg,Graduate Text in Mathematics, Vol 173,* ISBN 978-3-642-14278-9,pp.1-451,2010.

[2] C. Fragouli and E. Soljanin, “Network Coding Fundamentals”, *Foundation and Trends in Networking, Vol 2,* no.1, pp.1-133,2007.

[3] K. Menger, “Zur allgemeinen Kurventheorie,” *Fundamenta Mathematicae,*vol. 10, pp. 95–115, 1927.

[4] L. R. Ford Jr. and D. R. Fulkerson, “Maximal flow through a network,” *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.

[5] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, “Network information flow,” *IEEE Transactions on Information Theory*, vol. 46, pp. 1204–1216, July 2000.

[6] S.-Y. R. Li, R. W. Yeung, and N. Cai, “Linear network coding,” *IEEE Transactions on Information Theory*, vol. 49, pp. 371–381, February 2003.

[7] P. Sanders, S. Egner, and L. Tolhuizen, “Polynomial time algorithms for network information flow,” in *Proceedings of 15th ACM Symposium on Parallel Algorithms and Architectures*, 2003.

[8] S. Jaggi, P. A. Chou, and K. Jain, “Low complexity algebraic network multicast codes,” presented at *ISIT 2003*, Yokohama, Japan.

[9] S. Jaggi, P. Sanders, P. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, “Polynomial time algorithms for multicast network code construction,” *IEEE Transaction on Information Theory*, vol. 51, no. 6, no. 6, pp. 1973–1982, 2005.

[10] ´A. M. Barbero and Ø. Ytrehus, “Heuristic algorithms for small field multicast encoding,” *2006 IEEE International Symposium Information Theory (ISIT’06)*, Chengdu, China, October 2006.

[11] T. Ho, R. K¨otter, M. M´edard, M. Effros, J. Shi, and D. Karger, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory,* vol. 52, pp. 4413–4430, October 2006.

[12] C. Fragouli and E. Soljanin, "Information flow decomposition for network coding," *IEEE Transactions on Information Theory,* vol. 52, pp. 829–848, March 2006.

[13] K. Mehlhorn and P. Sanders "Algorithms and Data Structure", *Springer-Verlag,Heidelberg, ACM Computing Classification, Vol 173,* ISBN 978-3-540-77977-3,pp.1-300,2008.

4 EVOLUTIONARY APPROACH FOR NETWORK CODE CONSTRUCTION

The previous chapter has discussed the centralised and decentralised algorithms which apply to the network code construction. The focus of these is on network code construction without explicit attempts to save network and coding resources. This chapter describes a novel code construction method that overcomes issues with existing algorithms. It makes use of conflict multi - objective optimisation because traditional optimisation methods are unable to provide a reasonable solution. Thus evolutionary algorithms provide a significant method to solve the problem based on a GA with the following goals:

1. Minimise code design complexity;
2. Minimise network and coding resources;
3. Extend available network resources without dramatic network infrastructure alterations;
4. Design a protocol.

4.1 EVOLUTIONARY APPROACH AND EXPECTED ACHIEVEMENTS

The network code design algorithms in chapter 3 are fundamentally based on two network code design algorithms (LIF and Random Assignment) that do not contribute to identifying the associated minimal configuration or to the minimisation of network and coding resources. It is well known that the problem of finding the associated minimal configuration of a graph is NP-hard. Moreover, the problem of finding the minimum number of coding points is NP-hard for the majority of cases

[1]. Rather than tackling these NP-hard problems, the proposed method utilises an evolutionary approach to rapidly find a good solution in polynomial time.

4.1.1 MINIMISING CODE DESIGN COMPLEXITY

Code design complexity can be defined as the complexity of designing a feasible network coding scheme. This can be achieved by identifying the minimal configuration that simultaneously minimises network and coding resources during their multicast transmission.

Definition 4.1.1: Feasible Network Coding Scheme. An assignment of coding vectors is feasible if the coding vector of an edge *e* lies in the linear span of the coding vectors of the parent edges *In(e)*. A *valid* linear network code is any feasible assignment of coding vectors such that the matrix A_j is full rank for each receiver $t_j, 1 \le j \le N$.

Definition 4.1.1: Minimal Configuration. When the removal of one network edge causes at least one sink to lose its multicast properties, a configuration is a minimal configuration.

Figure 4-1 shows two multicast transmission minimal configurations for the network of Figure 3-1(b). That of Figure 4-1(a) consumes the more network and coding resources because transmission cost and coding resource usage (three coding nodes) are greater than in Figure 4-1(b). The former uses 19 links as opposed to the 18 in the latter. Thus the configuration of Figure 4-1(b) is more desirable but its identification is difficult even in this small example let alone in a large network leading to the proposed GA-based solution.

In the method proposed, every minimal configuration is identified by definition (4.1.1) satisfies the multicast properties meaning that coding nodes can be immediately assigned coding vectors safe in the knowledge that these satisfy the desired multicast properties. Unlike the LIF algorithm, coding nodes are not searched the topological order to assign the coding vectors rather the source can assign linearly independent coding vectors to the coding nodes in the identified minimal configuration and be guaranteed that all sinks simultaneously obtain full rank decoding matrixes.

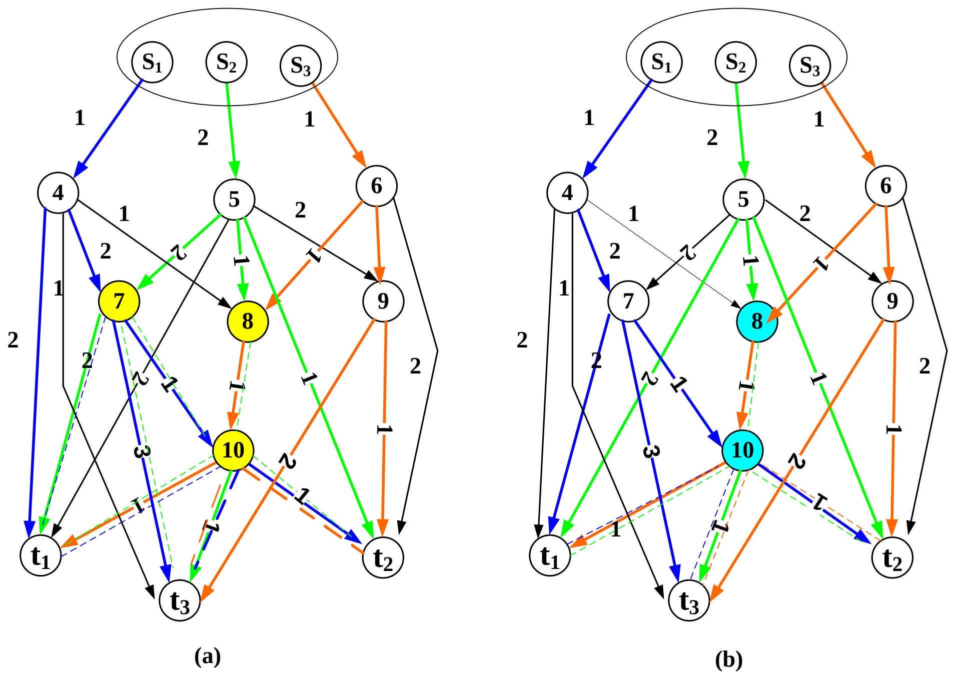


Figure 4-1: (a) Configuration of higher network and coding cost; (b) Configuration of lower network and coding cost.

4.1.2 MINIMISING THE NETWORK AND CODING RESOURCES

Network and coding resources are two vital factors during the codes construction process, and Chapter 5 considers coding resources and their optimisation in depth. The key intention of the proposed solution is to identify the minimal configurations with optimum network and coding resources from which the source selects one (e.g. Figure 4-1(b) above) for its code construction process.

4.1.3 MINIMALLY DISRUPTIVE AVAILABLE NETWORK RESOURCE EXTENSION

The established network infrastructure essentially consists of links and nodes. The former are constrained by factors such as bandwidth, cost, delay, availability and so on, the overcoming of which is assisted by network coding. The latter can be defined as (mostly inaccessible – e.g. satellite or undersea nodes) routers and consist of functionalities[4], computational power, hardware and software configurations and the like. Therefore substantial functional modifications are extremely costly and require consideration of hardware and software configurations. Nevertheless, they are capable of performing basic functionalities such as forwarding, duplicating, coding, decoding and basic mathematical operations which should be used in feasible solutions in the manner proposed here.

4.1.4 PROTOCOL DESIGN

This section discusses how the proposed solution contributes to design a network coding protocol. The previous sections explain that the proposed solution can identify the minimal configurations with their sparse matrices, shown for Figure 4-1 in Figure 4-2. A multicast network coding protocol can be developed based on the matrices to provide most essential requirements for network coding protocol design.

---

[4] A useful function within a computer application or program / The capacity of a computer program or application to provide a useful function

	S_1	S_2	S_3	4	5	6	7	8	9	10	t_1	t_2	t_3
S_1	0	0	0	1	0	0	0	0	0	0	0	0	0
S_2	0	0	0	0	1	0	0	0	0	0	0	0	0
S_3	0	0	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	1	0	0
5	0	0	0	0	0	0	1	1	0	0	0	1	0
6	0	0	0	0	0	0	0	1	1	0	0	0	0
7	0	0	0	0	0	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	0	0	0	1	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	1	1
10	0	0	0	0	0	0	0	0	0	0	1	1	1
t_1	0	0	0	0	0	0	0	0	0	0	0	0	0
t_2	0	0	0	0	0	0	0	0	0	0	0	0	0
t_3	0	0	0	0	0	0	0	0	0	0	0	0	0

(a)

	S_1	S_2	S_3	4	5	6	7	8	9	10	t_1	t_2	t_3
S_1	0	0	0	1	0	0	0	0	0	0	0	0	0
S_2	0	0	0	0	1	0	0	0	0	0	0	0	0
S_3	0	0	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	0	0	1	1	0
6	0	0	0	0	0	0	0	1	1	0	0	0	0
7	0	0	0	0	0	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	0	0	0	1	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	1	1
10	0	0	0	0	0	0	0	0	0	0	1	1	1
t_1	0	0	0	0	0	0	0	0	0	0	0	0	0
t_2	0	0	0	0	0	0	0	0	0	0	0	0	0
t_3	0	0	0	0	0	0	0	0	0	0	0	0	0

(b)

Figure 4-2: (a) and (b) Adjacent matrix representation for the minimal configuration of Figure 4-1(a) and (b) in order

4.1.4.1 Essential Requirements for Network Coding Protocol Design

1. Node discovery

   The previous section explained the minimal configuration and its sparse matrix representation. This section explains how source nodes, forwarding nodes, coding nodes and sinks are identified using the sparse matrix representations (MATLAB code in Appendix B-3).

   1.1 Source nodes

   If a node has zero input links and one or more out links, the node can be defined as the source node, and may be identified using the sparse matrix, when any matrix column consists of all zeros and its identical row consists of at least one "1" entry. In Figure 4-2 (a) the first three columns consist of all zeros and their identical rows consist of entries "1" at column indices {4, 5, 6} indentifying sources $\{S_1, S_2, S_3\}$.

1.2 Forwarding nodes

If a node has at least one input link and at least one output link, it is a forwarding node; when a forwarding node has more than one output link, it forwards identical copies of the original received packets. Forwarding nodes can be identified when any matrix column consists of at least one “1” entry and its identical row consists of at least one “1” entry. Nodes {4,5,6,7,8,9,10} in Figure 4-2 (a) can be identified as forwarding nodes.

1.3 Coding nodes

A coding node is a special type of forwarding node with more than one input link that forwards and linearly combines packets. Such nodes may be identified when any matrix column consists of more than one “1” entry, and its identical row consists of one or more “1” entries. Nodes {7,8,10} in Figure 4-2 (a) can be identified as the coding nodes.

1.4 Sink nodes

A node with one or more input links, and no output links, is a sink node that receives original source packets or their linear combinations via its input links. The sink node solves a linear system equation (Figure 3-12) to obtain the original source packets and is identifiable by a matrix column with one or more “1” entries and an identical row of all entries “0”. In Figure 4-2 (a), the last three columns or their identical rows are the sink nodes $\{t_1, t_2, t_3\}$.

2. Path selection

   Additional effort is not necessary for path selection because the proposed solution provides the source to sink minimal configuration with its sparse matrix comprising *hN* linear disjoint paths as described in Section 3.1.2.1.

3. Discovery of coding opportunities

   The proposed solution is able to identify the minimal configuration with optimum coding resources finding nodes via the sparse matrix (4.1.4.1-1.3 above).

4. Receiver selection

   The proposed solution commences with the two preliminary processes explained in Sections 3.1.2.1 and 3.1.2.2. These enable selection of the sink (identified as in Section 4.1.4.1-1.4) that is demanding the multicast data from the source.

5. Coding decision

   The coding decision is extremely difficult unless the coding opportunities have been discovered as in the proposed solution which delivers the minimal configuration with optimal coding opportunities (NP-hard).

4.2 NETWORK CODE CONSTRUCTION

This section discusses the network code construction using the sparse matrix, which is a representation of the minimal configuration with optimum network and coding resources.

	S_1	S_2	S_3	4	5	6	$\left\Vert\begin{matrix}\alpha_1[1\,0\,0]+\\ \alpha_2[0\,1\,0]\end{matrix}\right\Vert$ $7\Uparrow$	$\left\Vert\begin{matrix}\alpha_3[0\,1\,0]+\\ \alpha_4[0\,0\,1]\end{matrix}\right\Vert$ $8\Uparrow$	9	$\left\Vert\begin{matrix}\alpha_5[\alpha_1\;\alpha_2\;0]+\\ \alpha_6[0\;\alpha_3\;\alpha_4]\end{matrix}\right\Vert$ $10\Uparrow$	t_1	t_2	t_3
$[1\,0\,0]\Rightarrow S_1$	0	0	0	1	0	0	0	0	0	0	0	0	0
$[0\,1\,0]\Rightarrow S_2$	0	0	0	0	1	0	0	0	0	0	0	0	0
$[0\,0\,1]\Rightarrow S_3$	0	0	0	0	0	1	0	0	0	0	0	0	0
$[1\,0\,0]\rightarrow 4$	0	0	0	0	0	0	1	0	0	0	$\boxed{1}$	0	0
$[0\,1\,0]\rightarrow 5$	0	0	0	0	0	0	1	1	0	0	0	$\boxed{1}$	0
$[0\,0\,1]\rightarrow 6$	0	0	0	0	0	0	0	1	1	0	0	0	0
$[\alpha_1\,\alpha_2\,0] > 7$	0	0	0	0	0	0	0	0	0	1	$\boxed{1}$	0	$\boxed{1}$
$[0\,\alpha_3\,\alpha_4] > 8$	0	0	0	0	0	0	0	0	0	1	0	0	0
$[0\,0\,1]\rightarrow 9$	0	0	0	0	0	0	0	0	0	0	0	$\boxed{1}$	$\boxed{1}$
$\left\Vert\begin{matrix}\alpha_5[\alpha_1\;\alpha_2\;0]+\\ \alpha_6[0\;\alpha_3\;\alpha_4]\end{matrix}\right\Vert >> 10$	0	0	0	0	0	0	0	0	0	0	$\boxed{1}$	$\boxed{1}$	$\boxed{1}$
t_1	0	0	0	0	0	0	0	0	0	0	0	0	0
t_2	0	0	0	0	0	0	0	0	0	0	0	0	0
t_3	0	0	0	0	0	0	0	0	0	0	0	0	0

(*a*)

$$A_{t_1}=\begin{bmatrix}1 & 0 & 0\\ \alpha_1 & \alpha_2 & 0\\ \alpha_1\alpha_5 & (\alpha_2\alpha_5+\alpha_3\alpha_6) & \alpha_4\alpha_6\end{bmatrix}\quad A_{t_2}=\begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ \alpha_1\alpha_5 & (\alpha_2\alpha_5+\alpha_3\alpha_6) & \alpha_4\alpha_6\end{bmatrix}\quad A_{t_3}=\begin{bmatrix}\alpha_1 & \alpha_2 & 0\\ 0 & 0 & 1\\ \alpha_1\alpha_5 & (\alpha_2\alpha_5+\alpha_3\alpha_6) & \alpha_4\alpha_6\end{bmatrix}$$

(*b*)

$$A_{t_1}=\begin{bmatrix}1 & 0 & 0\\ 1 & 1 & 0\\ 1 & 0 & 1\end{bmatrix}\quad A_{t_2}=\begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 1 & 0 & 1\end{bmatrix}\quad A_{t_3}=\begin{bmatrix}1 & 1 & 0\\ 0 & 0 & 1\\ 1 & 0 & 1\end{bmatrix}$$

(*c*)

Figure 4-3: (a) Network codes construction using the sparse matrix; (b) Decoding matrixes for sinks {t_1, t_2, t_3}; (c) Decoding matrixes with finite field $\mathbb{F}_2$

The sparse matrix in Figure 4-2(a) is used to explain the network code construction process (NCCP). In section 4.1.4.1, the node discovery method identifies the source, forwarding, duplicating, coding and sink nodes using the sparse matrix. The source nodes are assigned source vectors, which are unit vectors $[1^{i-1}\ 0^{i}\quad 0^{h-i}]$, where the $(i-1)^{th}$ position of the vector is represented by the source node $(\{S_i\}, 1\le i\le h)$; this is set to 1 and other positions are zero. For example the vectors $[1\ 0\ 0], [0\ 1\ 0]$ and $[0\ 0\ 1]$ are assigned to the source nodes $\{S_1, S_2, S_3\}$ in Figure 4-3(a). Moreover, if any node duplicates or forwards original source data then it is assigned a related source vector. For example nodes $\{4, 5\}$ are assigned the source vectors $[1\ 0\ 0], [0\ 1\ 0]$ consecutively and nodes $\{6, 9\}$ are assigned the source vector $[0\ 0\ 1]$. Implementing the code assignment process requires $\mathrm{O}\left(n^2\right)$ operations for an *n*-sized sparse matrix.

Mutually-linear independent coding vectors are assigned to the coding nodes of the minimal configuration, since this independence is an essential requirement. Coding vector coefficients $\{\alpha_i\}$ are chosen in the finite field $\mathbb{F}_q$, and each in-link of the coding node is assigned a vector coefficient $\{\alpha_i\}$. Each incoming vector is multiplied by the relevant vector coefficient $\{\alpha_i\}$, and the resultants combined to obtain the coding vector. For example, the coding node {7} in Figure 4-3(a) has in-links from the forwarding nodes {4, 5}, and their source vectors are $[1\ 0\ 0], [0\ 1\ 0]$ in order. The coding coefficients $\{\alpha_1, \alpha_2\}$ are assigned to the in-links and the coding vector can be obtained by computing $\left[\!\left[\alpha_1 \left[1\ 0\ 0\right] + \alpha_2 \left[0\ 1\ 0\right]\right]\!\right]$. The coding node {8} can obtain its coding vector as the same as the node {7} but coding node {10} has in-links from {7, 8} and its coding vector is $\left[\!\left[\alpha_5 \left[\alpha_1\ \ \alpha_2\ \ 0\right] + \alpha_6 \left[0\ \alpha_3\ \ \alpha_4\right]\right]\!\right]$.

4.2.1 VALIDATING THE CODING VECTOR COEFFICIENTS $\{\alpha_i\}$

Definition 4.1.1 has laid out what is needed for a feasible network coding multicasting scheme in terms of the full rank of A_j for each receiver – the valid network coding vector coefficients $\{\alpha_i\}$ contribute to the formation of the full rank matrix.

Here the validation of the coding vector coefficients using the sparse matrix is described. Using the method presented in Section 1.4, sinks are identified. Figure 4-3 (b) shows the decoding matrixes $\{A_{t_1}, A_{t_2}, A_{t_3}\}$ for sinks $\{t_1, t_2, t_3\}$. For example, the sink *t₁* has unity entries in rows {4,7,10} and their vectors $\begin{bmatrix}1\ 0\ 0\end{bmatrix}$, $\begin{bmatrix}\alpha_1 \ \alpha_2 \ 0\end{bmatrix}$ and $\begin{bmatrix}\alpha_1\alpha_5 \ (\alpha_2\alpha_5 + \alpha_3\alpha_6) \ \alpha_4\alpha_6\end{bmatrix}$ form the decoding matrix A_{t_1}. Satisfying the condition: $f(\alpha_1, ...\alpha_\mu) = \det A_1 \det A_2 ......\det A_N \neq 0$ ensures that $\{\alpha_i\}$ are valid coefficients. In Figure 4-3 (b), the coefficients $\{\alpha_1, \dots, \alpha_6\}$ should be chosen over $\mathbb{F}_q$ satisfying $f(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6) = \det A_{t_1} \det A_{t_2} \det A_{t_3} \neq 0$. As seen in Figure 4-3 (c) $\mathbb{F}_2$ is sufficient as $f(1,1,1,1,1,1) = \det A_{t_1} \det A_{t_2} \det A_{t_3} \neq 0$.

This network code design algorithm can be defined as a centralised deterministic algorithm because it determines the minimal configuration with coding opportunities based on entire network information. The source provides the validated coding vector coefficients to their related coding points prior to initiating multicast transmission, and multicast packets are routed via the minimal configuration. The source can use the path selection method in Section 4.1.4.1-2 as a packet routing table.

The significant benefits of this algorithm are (a) the linear independency of the coding vectors can be tested to avoid decoding errors; (b) the size of finite field can be constrained. Moreover, since the proposed solution uses a GA at the source it requires only basic operations from simplified intermediate nodes.

To reduce computational complexity, *random linear network coding* can be introduced. This approach is *a partially centralised deterministic algorithm* because the minimal configuration with the coding opportunities is identified using the entire network information but codes are randomly assigned. Network codes are not validated and the network coding coefficients $\{\alpha_i\}$ are randomly chosen in an adequate large finite field $\mathbb{F}_q$. The source forwards its multicast data using the path selection method in Section 4.1.4.1-2 and the coding nodes of the minimal configuration linearly combine their incoming packets using the random coefficients $\{\alpha_i^R\}$.

This random approach does mean that there is a higher probability of linear dependency which will affect the performance of the system, the finite field size (|q|) should be large enough, affecting the computational and network coding resources; the functional integration of the coding nodes is essential to a random number generation. Although most operating systems can provide “random” number generators but the resulting numbers are not always sufficiently random. This may be avoided by employing a number generator that has been shown to have acceptable performance [2], such as the Mersenne Twister [3].

4.3 EVOLUTIONARY ALGORITHMS

Evolutionary Algorithms (EAs) are heuristic methods that solve combinatorial optimisation problems. They originate in Darwin's theory of evolution [6], which introduced three fundamental components of evolution: replication, variation and natural selection. The first is the formation of a new organism from a previous one such that errors arise, known as variations (such as sexual reproduction) to allow evolutionary changes to occur. Natural selection taking place when individuals compete for scarce environmental resources and reproduction opportunities allows the fittest individuals at the expense of the weakest. GAs are a type of evolutionary algorithm inspired by the evolutionist theory explaining the origin of species. In nature, weak and unfit species within their environment are faced with extinction by natural selection. The strong ones have greater opportunity to pass their genes to future generations via reproduction. In the long run, species carrying the correct combination in their genes become dominant in their population. Sometimes, during the slow process of evolution, random changes may occur in genes. If these changes provide additional advantages in the challenge for survival, new species evolve from the old ones. Unsuccessful changes are eliminated by natural selection.

4.3.1 GENETIC ALGORITHM

In GA terminology, a solution vector $x \in X$ is called an individual or a *chromosome* made of discrete units called *genes,* each of which controls one or more features of the chromosome. In the original implementation of GA by Holland [6], genes are assumed to be binary digits whereas in later implementations, more varied gene types have been introduced [11]. Normally, a chromosome corresponds to a unique solution *x* in the solution space; this requires a mapping mechanism between solution space and chromosomes known as an *encoding* - GAs work on the encoding of a problem, not on the problem itself.

GAs operate with a collection of chromosomes, called a *population* which is normally randomly initialized. As the search evolves, the population includes fitter and fitter solutions, and eventually it converges (dominated by a single solution). Two operators are used to generate new solutions from existing ones: *crossover* and *mutation*. The first is the most important in which generally two chromosomes, called *parents*, are combined together to form new chromosomes, called *offspring*. The parents are selected from existing chromosomes in the population with preference towards fitness so that offspring are expected to inherit good genes which make the parents fitter. By iteratively applying the crossover operator, genes of good chromosomes are expected to appear more frequently in the population, eventually leading to convergence to an overall good solution.

Mutation introduces random changes into the chromosome characteristics, and is generally applied at the gene level. In typical GA implementations, the mutation rate (probability of gene property change) is very small and depends on the chromosome length. Therefore, the new chromosome produced by mutation will not be very different from the original one. Nevertheless, mutation is crucial in GAs because crossover produces population convergence so mutation reintroduces genetic diversity assisting in escaping from local optima.

Reproduction involves selection of chromosomes for the next generation. In the most general case, the fitness of an individual determines the probability of its survival for the next generation. There are different selection procedures in a GA depending on how the fitness values are used. Proportional selection, ranking, and tournament selection are the most popular selection procedures. The procedure of a generic GA [7] is given in Figure 4-4.

```
Input:     $K$        (population size)
           $T$        (maximum number of generations)
           $p_c$      (crossover probability)
           $p_m$      (mutation rate)
Output:    $\mathbf{A}$        (nondominated set)

Step 1:    Initialization: Set $\mathbf{P}_0 = \emptyset$ and $t = 0$. For $i = 1,\ldots\ldots K$ do
             a) Choose $\mathbf{i} \in \mathbf{I}$ according to some probability distribution /
                randomly.
             b) Set $\mathbf{P}_0 = \mathbf{P}_0 + \{\mathbf{i}\}$.

Step 2:    Fitness assignment: For each individual $\mathbf{i} \in \mathbf{P}_t$ determine the encoded
           decision vector $\mathbf{x} = \mathbf{m}(i)$ as well as the objective vector $\mathbf{y} = \mathbf{f(x)}$ and
           calculate the scalar fitness value $F(\mathbf{i})$.

Step 3:    Selection: Set $\mathbf{P'} = \emptyset$. For $i = 1,\ldots\ldots K$ do
             a) Select one individual $\mathbf{i} \in \mathbf{P}_t$ according to a given scheme and
                based on its fitness value $F(\mathbf{i})$.
             b) Set $\mathbf{P'} = \mathbf{P'} + \{\mathbf{i}\}$.
                The temporary population $\mathbf{P'}$ is called the mating pool.

Step 4:    Recombination: Set $\mathbf{P''} = \emptyset$. For $i = 1,\ldots\ldots K/2$ do
           a) Choose two individuals $\mathbf{i}, \mathbf{j} \in \mathbf{P'}$ and remove them from $\mathbf{P'}$.
           b) Recombine $\mathbf{i}$ and $\mathbf{j}$ . The resulting children are $\mathbf{k}, \mathbf{l} \in \mathbf{I}$ .
           c) Add $\mathbf{k}, \mathbf{l}$ to $\mathbf{P''}$ with probability $p_c$. Otherwise add $\mathbf{i}, \mathbf{j}$ to $\mathbf{P''}$.

Step 5:    Mutation: Set $\mathbf{P'''} = \emptyset$. For each individual $\mathbf{i} = \mathbf{P''}$ do
           a) Mutate $\mathbf{i}$ with mutation rate $p_m$. The resulting individual is $\mathbf{j} \in \mathbf{I}$ .
           b) Set $\mathbf{P'''} = \mathbf{P'''} + \{\mathbf{j}\}$.

Step 6:    Termination: Set $\mathbf{P}_{t+1} = \mathbf{P'''}$ and $t = t + 1$. If $t \geq T$ or another
           stopping criterion is satisfied then set $\mathbf{A} = p(\mathbf{m}(\mathbf{P}_t))$ else go to Step 2.
```

Figure 4-4: Generic GA procedure

In the selection process, which can be either stochastic or completely deterministic, low-quality individuals are removed from the population, while high quality individuals are reproduced. The goal is to focus the search on particular portions of the search space and to increase the average quality within the population. The quality of an individual with respect to the optimization task is represented by a scalar value, the so-called *fitness*. Note that since the quality is related to the objective functions and the constraints, an individual must first be decoded before its fitness can be calculated. This situation is illustrated in Figure 4-5. Given an individual $\boldsymbol{i} \in \boldsymbol{I}$. A mapping function $\boldsymbol{m}$ encapsulates the decoding algorithm to derive the decision vector $\boldsymbol{x} = \boldsymbol{m}(\boldsymbol{i})$ from $\boldsymbol{i}$. Applying $\boldsymbol{f}$ to $\boldsymbol{x}$ yields the corresponding objective vector on the basis of which a fitness value is assigned to $\boldsymbol{i}$.

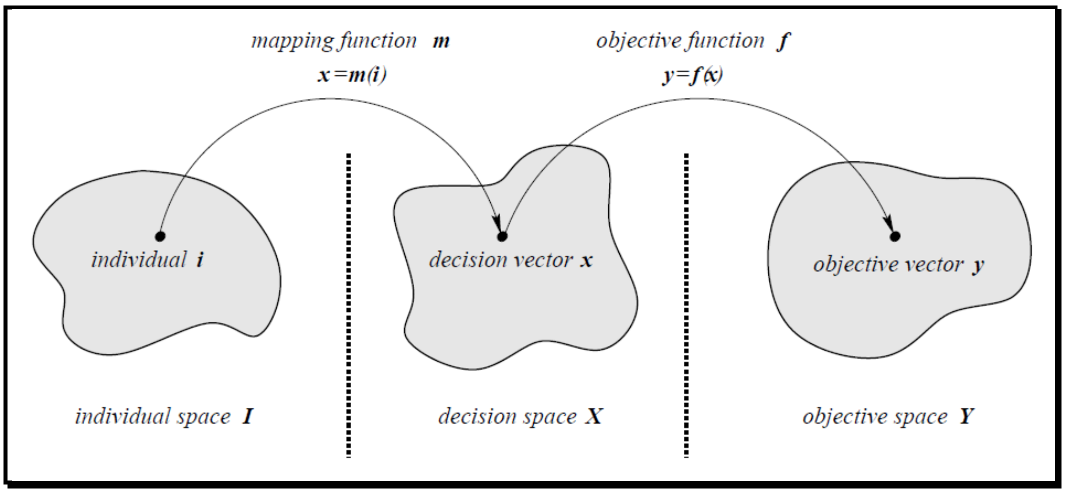


Figure 4-5: Relation between individual space, decision space, and objective space.

4.3.2 MULTI-OBJECTIVE GAS

Being population-based approaches, GAs are well suited to solve multi-objective optimization problems. A generic single-objective GA can be modified to find a set of multiple non-dominated solutions in a single run. The ability of a GA to simultaneously search different regions of a solution space makes it possible to find a diverse set of solutions for difficult problems with non-convex, discontinuous, and multi-modal solutions spaces. The first multi-objective GA, called vector evaluated GA (or VEGA), was proposed by Schaffer [9]. Afterwards, several multi-objective evolutionary algorithms were developed including the Multi-objective Genetic Algorithm (MOGA) [10]. Here customised algorithms are designed for multicast NC by adapting strategies from VEGA and MOGA.

4.3.2.1 Multi-Objective Genetic Algorithm (MOGA)

A generic single-objective GA in Figure 4-4 is modified to find a set of multiple non-dominated solutions in a single run. Enhancing the potential of a GA to simultaneously search different region of a solution space, MOGA is a promising candidate to find a diverse solution set for difficult (e.g. non-convex) problems. The GA crossover operator exploits structures of good solutions with respect to different objectives to create new non-dominated solutions in unexplored parts of a Pareto front.

4.3.2.2 Vector-Evaluated Genetic Algorithm (VEGA)

In this method, the GA selection operator is modified, so that at each generation, a number of sub-populations is generated by performing proportional selection according to each objective function in turn. Thus, for a population size K and number of objectives q, each sub-population’s size is K/q. These sub-populations are shuffled together to obtain a new population of size K, and new generations created by the usual GA operations as shown in Figure 4-6.

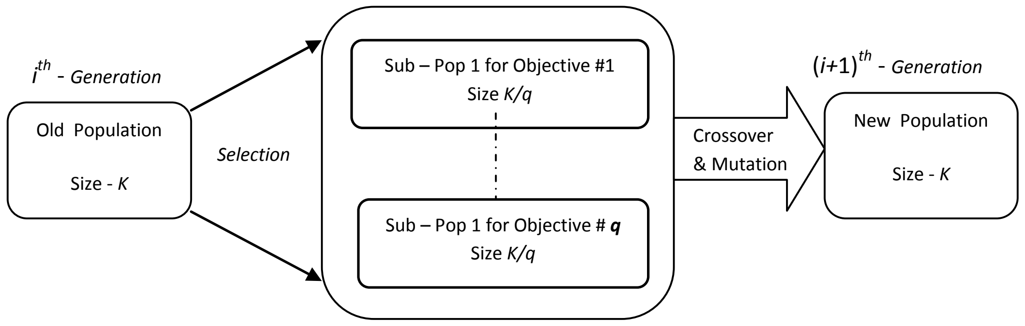


Figure 4-6: VEGA procedure

4.3.3 MULTI-OBJECTIVE OPTIMIZATION USING GENETIC ALGORITHMS

Multi-objective formulations are realistic models for many complex engineering optimization problems. In many real-life problems, the objectives under consideration conflict with each other (e.g. minimize cost, maximize performance, maximize reliability) so optimisation with respect to a single objective can result in unacceptable results with respect to the other objectives. A reasonable solution to a multi-objective problem is to investigate a set of solutions, each of which satisfies the objectives at an acceptable level without being dominated by any other solution. This section presents GAs developed specifically for problems with multiple objectives that utilise special fitness functions and methods to promote solution diversity.

4.3.3.1 Single-objective optimisation formulation

The optimization problems are normally stated in a single-objective way. In other words, the process must optimise a single objective function complying with a series of constraints.

A single-objective optimisation problem may be stated as follows:

Optimise [minimise/maximise]

Function $f(X)$

Subject to

Functions of constraints

$$H(X) = 0$$

$$G(X) \leq 0$$

For this problem three sets of solutions can be defined:

1. The *universal set,* which in this case is all possible values of X , whether feasible or not.
2. The *set of feasible solutions,* which are all the values of X that comply with the constraints.
3. The *set of optimal solutions,* which are those values of X that, in addition to being feasible, comply with the optimal value of function $f(X)$, whether in a specific $[a,b]$ interval (local optimal solutions) or in a global context $[\infty^{-}, \infty^{+}]$. In this case, one says that the set of optimal solutions may consists of a single element or several elements, provided that the following characteristic is met: $f(x) = f(x')$, where $x \neq x'$. In this case, we can say that there are two optimal values to the problem when vector $X = \{x, x'\}$.

4.3.3.2 Multi-objective optimisation formulation

Consider a decision-maker who wishes to optimize ***q*** objectives such that the objectives are non-commensurable and the decision-maker has no clear preference for the objectives relative to each other. Without loss of generality, all objectives are of the minimization type since this can be converted to a maximization type by multiplying by minus one. A minimization multi-objective decision problem with ***q*** objectives is defined as follows: Given an *n*-dimensional decision variable vector $\vec{x} = \{x_1, x_2, ...x_n\}$ in the solution space X , find a vector $\overrightarrow{x^*}$ that minimizes a given set of ***q*** objective functions $z(\overrightarrow{x^*}) = \{z_1(\overrightarrow{x^*}), z_2(\overrightarrow{x^*}), ...z_q(\overrightarrow{x^*})\}$. The solution space X is generally restricted by a series of constraints, such as $g(\overrightarrow{x^*}) = b_j$ for $j = 1,....m$, and bounds on the decision variables.

If all objective functions are for minimization, a feasible solution $\vec{x}$ is said to dominate another feasible solution $\vec{y}$ $(\vec{x} \succ \vec{y})$, if and only if, $z_i(\vec{x}) \leq z_i(\vec{y})$ for $j = 1,....K$ and $z_j(\vec{x}) < z_j(\vec{y})$ for least one objective function *j*. A solution is said to be *Pareto optimal* if it is not dominated by any other solution in the solution space. A Pareto optimal solution cannot be improved with respect to any objective without worsening at least one other objective. The set of all feasible non-dominated solutions in X is referred to as the Pareto optimal set, and for a given Pareto optimal set, the corresponding objective function values in the objective space are called the *Pareto front*. For many problems, the number of Pareto optimal solutions is very large (perhaps infinite).

The ultimate goal of a multi-objective optimization algorithm is to identify solutions in the Pareto optimal set. However, identifying the entire Pareto optimal set is practically impossible for many multi-objective problems due to its size. In addition, for many problems, especially for combinatorial optimization problems, proof of solution optimality is computationally infeasible. Therefore, a practical approach to multi-objective optimization is to investigate a set of solutions (the best-known Pareto set) that represent the Pareto optimal set as well as possible. With these concerns in mind, a multi-objective optimization approach should achieve the following three conflicting goals [8]:

1. The best-known Pareto front should be as close as possible to the true Pareto front. Ideally, the best-known Pareto set should be a subset of the Pareto optimal set.
2. Solutions in the best-known Pareto set should be uniformly distributed and diverse over of the Pareto front in order to provide the decision-maker a true picture of trade-offs.
3. The best-known Pareto front should capture the whole spectrum of the Pareto front. This requires investigating solutions at the extreme ends of the objective function space.

For a given computational time limit, the first goal is best served by focusing (intensifying) the search on a particular region of the Pareto front. On the contrary, the second goal demands the search effort to be uniformly distributed over the Pareto front. The third goal aims at extending the Pareto front at both ends, exploring new extreme solutions.

4.4 EVOLUTIONARY APPROACH FOR IDENTIFY THE MINIMAL CONFIGURATIONS

Identify the minimal configuration with optimum network and coding resources is extremely difficult. The optimisation problem to find the minimal number of required coding nodes is NP-hard [4]. Even approximating the minimal number of coding nodes within any multiplicative factor, or within an additive factor of $|V|^{1-\varepsilon}$, is NP-hard [5]. The evolutionary approach based on a genetic algorithm provides solutions to avoid the computational complexity that makes the problem NP-hard. There now follows a discussion of how the evolutionary algorithm based on a multi-objective GA is used to identify the minimal configurations with optimum network and coding resources. Figure 4-7 shows a block diagram of the process which comprises two fundamental processes: the preliminary process (which creates a search space) and the multi-objective GA process itself.

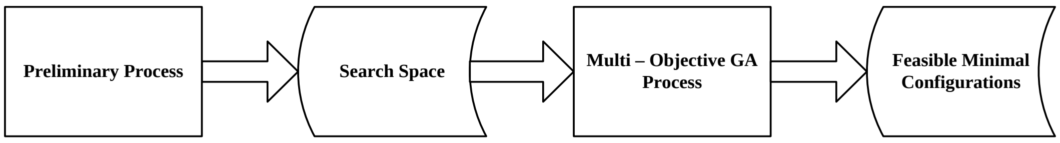


Figure 4-7: Solution phase for identifying the feasible minimal configurations

4.4.1 PRELIMINARY PROCESS

The preliminary process provides unevaluated individuals to the search space and then the two generic algorithms (path augmenting and linear disjoint path –see Section 3.1.2) contribute to create the search space. Figure 3-10 shows all available linear disjoint paths from sources $\{S_1, S_2, S_3\}$ to receivers $\{t_1, t_2, t_3\}$ for the acyclic graph in Figure 3-1(b). Here, Figure 4-8 shows sets of 3-linear disjoint paths and which are classified based on sink IDs. These sets are nominated as $Gn_{t_j}^{x}$ where $(1 \le j \le N)$ and x is undefined number. For example, sink – t_1 has three sets of 3 – linear disjoint paths, which are: $Gn_{t_1}^{1}, Gn_{t_1}^{2}$ and $Gn_{t_1}^{3}$.

$$\left\{\begin{array}{ccccc}
t_1 & & t_2 & & t_3\\
\Downarrow & & \Downarrow & & \Downarrow\\
\left\{\begin{array}{l}\langle S_1,4,t_1\rangle\\ \langle S_2,5,7,t_1\rangle\\ \langle S_3,6,8,10,t_1\rangle\end{array}\right\} & \rightarrow & \left\{\begin{array}{l}\langle S_1,4,7,10,t_2\rangle\\ \langle S_2,5,t_2\rangle\\ \langle S_3,6,9,t_2\rangle\end{array}\right\} & \rightarrow & \left\{\begin{array}{l}\langle S_1,4,t_3\rangle\\ \langle S_2,5,8,10,t_3\rangle\\ \langle S_3,6,9,t_3\rangle\end{array}\right\}\\
& \searrow & & \searrow & \\
\left\{\begin{array}{l}\langle S_1,4,t_1\rangle\\ \langle S_2,5,t_1\rangle\\ \langle S_3,6,8,10,t_1\rangle\end{array}\right\} & & \left\{\begin{array}{l}\langle S_1,4,8,10,t_2\rangle\\ \langle S_2,5,t_2\rangle\\ \langle S_3,6,9,t_2\rangle\end{array}\right\} & & \left\{\begin{array}{l}\langle S_1,4,t_3\rangle\\ \langle S_2,5,7,t_3\rangle\\ \langle S_3,6,9,t_3\rangle\end{array}\right\}\\
\left\{\begin{array}{l}\langle S_1,4,7,t_1\rangle\\ \langle S_2,5,t_1\rangle\\ \langle S_3,6,8,10,t_1\rangle\end{array}\right\} & & & & \left\{\begin{array}{l}\langle S_1,4,t_3\rangle\\ \langle S_2,5,7,10,t_3\rangle\\ \langle S_3,6,9,t_3\rangle\end{array}\right\}\\
& & & & \left\{\begin{array}{l}\langle S_1,4,t_3\rangle\\ \langle S_2,5,7,t_3\rangle\\ \langle S_3,6,8,10,t_3\rangle\end{array}\right\}\\
& & & & \left\{\begin{array}{l}\langle S_1,4,7,t_3\rangle\\ \langle S_2,5,8,10,t_3\rangle\\ \langle S_3,6,9,t_3\rangle\end{array}\right\}
\end{array}\right\}
\Rightarrow
\begin{array}{c}
\begin{array}{ccc} t_1 & t_2 & t_3\\ \Downarrow & \Downarrow & \Downarrow\end{array}\\
\left\{\begin{array}{ccc}
Gn^{1}_{t1} & Gn^{1}_{t2} & Gn^{1}_{t3}\\
Gn^{2}_{t1} & Gn^{2}_{t2} & Gn^{2}_{t3}\\
Gn^{3}_{t1} & & Gn^{3}_{t3}\\
 & & Gn^{4}_{t3}\\
 & & Gn^{5}_{t3}
\end{array}\right\}
\end{array}$$

Figure 4-8: All available sets of 3 - linear disjoint paths for receivers {t_1, t_2, t_3}

Based on the sets of 3-linear disjoint paths in Figure 4-8, a search space creation process is shown in Figure 4-9. A random shuffled process picks $Gn_{t_j}^{x}$ from each sink column $t_j, (1 \le j \le N)$ and creates a row. A row is defined as an *individual* and its elements $Gn_{t_j}^{x}$ are defined as *genes*. The random shuffled process is terminated when a size of the search space (Z) reaches a pre defined number. This method is computationally efficient; while it may cause identical individuals this is not a significant issue because the proposed solution is applied to analyse a large scale network with a large number of sinks. Moreover this is a significant stage of the proposed solution because it is a commencement to map the network coding problem into a GA framework.

The search space of the problem is not smooth or unimodal (all objective constraints are unknown) with respect to the number of sets of linear disjoint paths because each sink has different combination sets of the linear disjoint paths. The search space in this work consists of a large number of feasible or infeasible individuals which are created by the different combination sets of linear disjoint paths. An NP-hard problem results in which the individuals are not well understood and it is not critical that the calculated solution may not be a global optimum. It should also be noted that, while it is hard to characterize the structure of the search space, once provided with a solution we can verify its feasibility (calculating three objective functions in section 4.4.2.1) in polynomial time. Thus, if the use of genetic operations can suitably limit the size of the space to be actually searched allowing a solution to be obtained relatively efficiently.

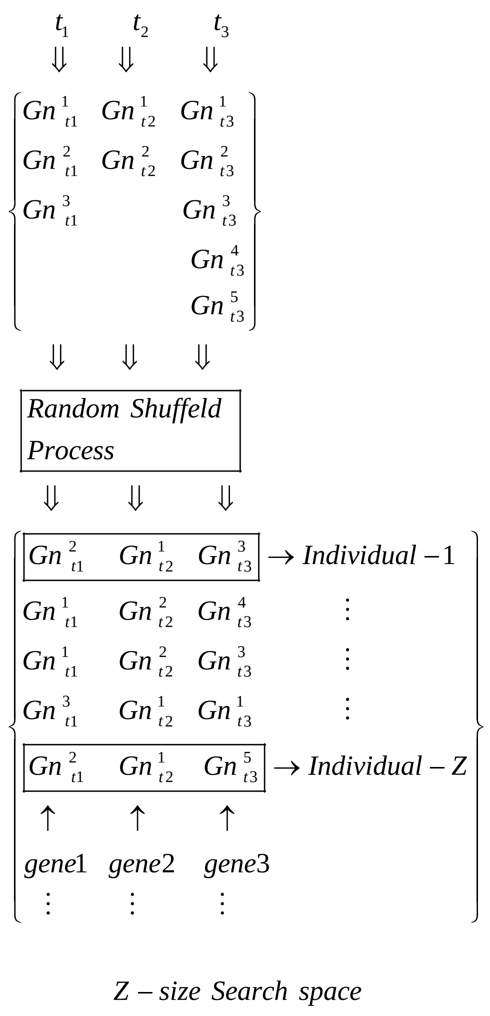

t_1 t_2 t_3
Gn^1_{t1} Gn^1_{t2} Gn^1_{t3}
Gn^2_{t1} Gn^2_{t2} Gn^2_{t3}
Gn^3_{t1} Gn^3_{t3}
Gn^4_{t3}
Gn^5_{t3}
Random Shuffeld Process
Gn^2_{t1} Gn^1_{t2} Gn^3_{t3} → Individual − 1
Gn^1_{t1} Gn^2_{t2} Gn^4_{t3}
Gn^1_{t1} Gn^2_{t2} Gn^3_{t3}
Gn^3_{t1} Gn^1_{t2} Gn^1_{t3}
Gn^2_{t1} Gn^1_{t2} Gn^5_{t3} → Individual − Z
gene1 gene2 gene3
Z − size Search space

Figure 4-9: The search space creation process

4.4.2 MULTI-OBJECTIVE GA PROCESS

This part is vital to identify feasible configurations and section concerns how MOGA and VEGA perform on the search space evaluated by means of simulation.

4.4.2.1 Fitness Assignment and Individual Evaluation $F_I = \{f_I(X_i), f_I(Y_j), f_I(Z_k)\}$

The individuals in the initial population or mating pool are assigned their fitness following the objective functions. Three objective functions are presented below to optimise the three major factors in multicast transmission with network coding: network resources, network cost and coding resources.

1. Number of coding nodes in the individual - *I*: $f_I(X_i)$
2. Number of hops (edges) in the individual - *I*: $f_I(Y_j)$
3. Total hop distances in the individual - *I*: $f_I(Z_k)$

Since the usage of network coding resources depends on the number of coding nodes, $f_I(X_i)$ optimises this aspect. Here, the aim is to identify the minimal configuration rather than single paths between the source and sinks, making shortest path identification essential but not an overriding concern of the process. Thus $f_I(Y_j)$ provides a way to optimise the number of edges in the minimal configuration, which is a strategic technique to identify the shortest paths but avoids an excessive number of constraints and objective functions that would prohibitively increase the computational complexity. Furthermore, $f_I(Z_k)$ encourages the shortest paths in the minimal configuration and consequently both $f_I(Y_j)$ and $f_I(Z_k)$ contribute to optimise the multicast network cost.

The objective functions are evaluated for each individual (I) using the sparse matrix shown in Figure 4-3(a) for the minimal configuration in Figure 4-1(a). Section 4.1.4.1-1.3 explains the method to identify the coding nodes in the minimal configuration using the sparse matrix, and this provides an excellent way to evaluate f_I (X_i). Moreover, the '1' entries of the sparse matrix represent the hops of the minimal configuration with f_I (Y_j) being just the sum of all the '1' entries. Section 3.1.1.2 discusses the adjacent matrix representation for the multicast DAG and its weight matrix. In this optimisation, the distances of the hops are used to create the weight matrix. The evolutionary approach identifies the feasible minimal configuration as the sparse matrix and its elements are compared with the distance matrix of the entire network to obtain the distance matrix of the feasible configuration. For a example, Figure 4-10 shows a distance matrix for the minimal configuration in Figure 4-1(a) $f_I(Z_k)$ is evaluated by Equation 4-1.

$$f_I(Z_k) = \sum d_{ij} \qquad \textbf{Equation 4-1}$$

Where d_{ij} – a distance of hop (i,j)

$$\begin{array}{c|ccccccccccccc}
 & S_1 & S_2 & S_3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & t_1 & t_2 & t_3 \\ \hline
S_1 & 0 & 0 & 0 & d_{S_1,4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_2 & 0 & 0 & 0 & 0 & d_{S_2,5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_3 & 0 & 0 & 0 & 0 & 0 & d_{S_3,6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & d_{4,7} & 0 & 0 & 0 & d_{4,t_1} & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & d_{5,7} & d_{5,8} & 0 & 0 & 0 & d_{5,t_2} & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d_{6,8} & d_{6,9} & 0 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d_{7,10} & d_{7,t_1} & 0 & d_{7,t_3} \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d_{8,10} & 0 & 0 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d_{9,t_2} & d_{9,t_3} \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d_{10,t_1} & d_{10,t_2} & d_{10,t_3} \\
t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Figure 4-10: Distance matrix for the minimal configuration in Figure 4-1(a)

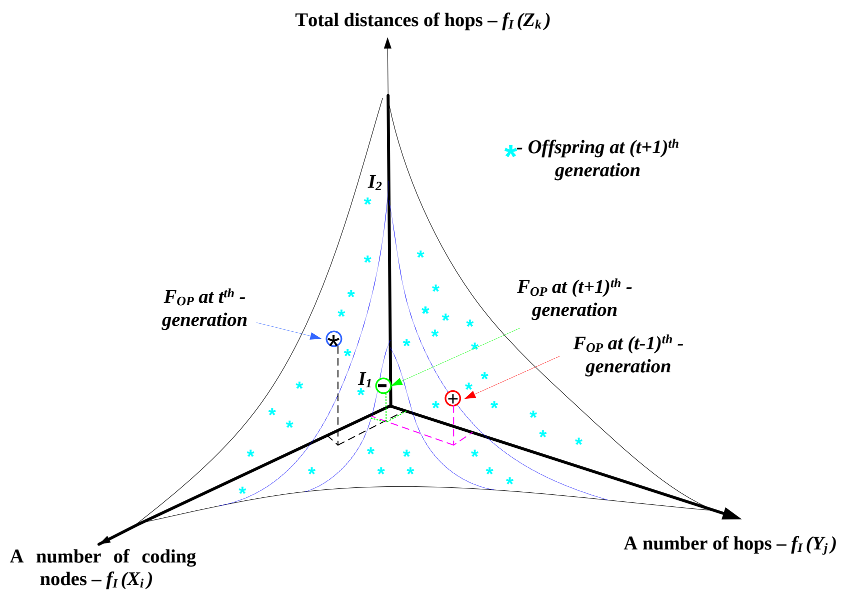
The problem is thus converted to a multi-objective optimisation problem and in such problems the objectives are generally in conflict. This is the case here since when the number of hops is optimised via $f_I(Y_j)$ it is likely that most of the coding nodes will be removed, increasing total hop distances in $f_I(Z_k)$.

Here, the traditional GA is customised to accommodate multi-objective problems by using specialised fitness functions and introducing method to promote solution diversity. The approach is to determine an entire Pareto optimal solution set which is a highly suitable approach because all three objectives do not have pre-identified constraints. Therefore, the Pareto optimal front is updated at the end of each generation by comparing it with that of the previous generation. For example, in Figure 4-11, the Pareto optimal front of the *(t-1)*th – generation is: $F_{OP}^{t-1} = \{f_{OP}(X_i^{t-1}) \neq 0, f_{OP}(Y_j^{t-1}) \neq 0, f_{OP}(Z_k^{t-1}) \neq 0\}$ and of the *t*th–generation is: $F_{OP}^{t} = \{f_{OP}(X_{i'}^{t}) \neq 0, f_{OP}(Y_{j'}^{t}) \neq 0, f_{OP}(Z_{k'}^{t}) \neq 0\}$. Assuming that if $f_{OP}(X_i^{t-1}) < f_{OP}(X_{i'}^{t})$, $f_{OP}(Y_j^{t-1}) > f_{OP}(Y_{j'}^{t})$ and $f_{OP}(Z_k^{t-1}) < f_{OP}(Z_{k'}^{t})$ then the Pareto optimal front of the (*t+1*)th – generation $F_{OP}^{t+1} = \{f_{OP}(X_i^{t+1}), f_{OP}(Y_j^{t+1}), f_{OP}(Z_k^{t+1})\}$ is obtained as below.

$$f_{OP}(X_i^{t+1}) \Leftarrow f_{OP}(X_i^{t-1})$$

$$f_{OP}(Y_j^{t+1}) \Leftarrow f_{OP}(Y_{j'}^{t})$$

$$f_{OP}(Z_k^{t+1}) \Leftarrow f_{OP}(Z_k^{t-1})$$



Total distances of hops – $f_I(Z_k)$
- Offspring at (t+1)th generation
I_2
F_{OP} at tth - generation
F_{OP} at (t+1)th - generation
F_{OP} at (t-1)th - generation
I_1
A number of coding nodes – $f_I(X_i)$
A number of hops – $f_I(Y_j)$

Figure 4-11: (*t*+1)th – generation evaluation and obtain Pareto optimal (F_{OP}) for (*t*+1)th – generation

The mutual comparisons between individuals are extremely hard in multi-objective optimisation and the proposed method avoids this difficulty because each individual of *t*th generation is compared with the Pareto optimal (F_{OP}^{t}) of the *t*th generation. In the third step of the generic GA in Figure 4-4, each individual (I) is assigned its fitness $F_I = \{f_I(X_i), f_I(Y_j), f_I(Z_k)\}$ using the objective functions and it is compared with the Pareto optimum of its generation. Any individual far away from this is defined as a less fit or infeasible individual (e.g. I_2 in Figure 4-11) and vice versa (e.g. I_1 in Figure 4-11 which is to be preferred).

4.4.2.2 GA Operations on the Search Space

Section 4.4.1 has discussed the preliminary process in detail and the optimisation and searching will now be covered. An initial population ($P_{t=1}$) is obtained by randomly picking individuals in the search space at $t=1$. The GA operations operate on P_1 to form a new generation (P_2). The population size (K) is constantly maintained throughout the GA operations and is equal to the size of the initial population. The generation (P_t) is evaluated by the evaluation process in section 4.4.2.1 and highest fitness individuals are recombined by crossover to form an offspring population (Q_t). Mutation of this population ensures that individuals identical to their parents do not occur, so that generation P_t shows a significant diversion from previous generation P_{t-1}. The iteration ends when the size of Q_t is equal to K, and Q_t is assigned to generation P_t, which is evaluated to find the feasible minimal configurations; this process repeats as long as a termination criterion is not satisfied.

4.4.2.2.1 Crossover

Each individual created as in Section 4.4.1 is comprised of genes that can be denoted by a unique sink-ID so the size (*N*) of the individual is bounded by the number of sinks.

In the simulation, *single point crossover* is employed with the crossover point β, selected as follows for crossover probability pr_c:

$$\beta = \begin{cases} \lfloor N(pr_c) \rfloor & \{N(pr_c)\} < 0.5 \\ \lceil N(pr_c) \rceil & \{N(pr_c)\} \geq 0.5 \end{cases}$$

Where $\{N(pr_c)\}$ denotes the fractional part of $N(pr_c)$. A value of 0.7 for pr_c was found to give good results after experimentation. Figure 4-12 shows the crossover operation at a gene level and the crossover point β is calculated as 2 (3x0.7= 2.1, fraction 0.1<0.5 then $\beta = 2$).

$$\mathit{Single\ Point\ Crossover} \Downarrow$$

$$\left[\!\left[\left\{\begin{matrix} \langle S_1,4,t_1\rangle \\ \langle S_2,5,7,t_1\rangle \\ \langle S_3,6,8,10,t_1\rangle \end{matrix}\right\} \left\{\begin{matrix} \langle S_1,4,7,10,t_2\rangle \\ \langle S_2,5,t_2\rangle \\ \langle S_3,6,9,t_2\rangle \end{matrix}\right\} \left\{\begin{matrix} \langle S_1,4,t_3\rangle \\ \langle S_2,5,8,10,t_3\rangle \\ \langle S_3,6,9,t_3\rangle \end{matrix}\right\} \right]\!\right] \Rightarrow \left[\!\left[\left\{\begin{matrix} \langle S_1,4,t_1\rangle \\ \langle S_2,5,7,t_1\rangle \\ \langle S_3,6,8,10,t_1\rangle \end{matrix}\right\} \left\{\begin{matrix} \langle S_1,4,7,10,t_2\rangle \\ \langle S_2,5,t_2\rangle \\ \langle S_3,6,9,t_2\rangle \end{matrix}\right\} \left\{\begin{matrix} \langle S_1,4,t_3\rangle \\ \langle S_2,5,7,t_3\rangle \\ \langle S_3,6,9,t_3\rangle \end{matrix}\right\} \right]\!\right]$$

$$F_{I-1} = \{3,19,26\} \quad \Downarrow \quad \Uparrow \qquad\qquad F_{I'-1} = \{2,18,29\}$$

$$\left[\!\left[\left\{\begin{matrix} \langle S_1,4,t_1\rangle \\ \langle S_2,5,t_1\rangle \\ \langle S_3,6,8,10,t_1\rangle \end{matrix}\right\} \left\{\begin{matrix} \langle S_1,4,7,10,t_2\rangle \\ \langle S_2,5,t_2\rangle \\ \langle S_3,6,9,t_2\rangle \end{matrix}\right\} \left\{\begin{matrix} \langle S_1,4,t_3\rangle \\ \langle S_2,5,7,t_3\rangle \\ \langle S_3,6,9,t_3\rangle \end{matrix}\right\} \right]\!\right] \Rightarrow \left[\!\left[\left\{\begin{matrix} \langle S_1,4,t_1\rangle \\ \langle S_2,5,t_1\rangle \\ \langle S_3,6,8,10,t_1\rangle \end{matrix}\right\} \left\{\begin{matrix} \langle S_1,4,7,10,t_2\rangle \\ \langle S_2,5,t_2\rangle \\ \langle S_3,6,9,t_2\rangle \end{matrix}\right\} \left\{\begin{matrix} \langle S_1,4,t_3\rangle \\ \langle S_2,5,8,10,t_3\rangle \\ \langle S_3,6,9,t_3\rangle \end{matrix}\right\} \right]\!\right]$$

$$F_{I-2} = \{2,18,27\} \qquad\qquad F_{I'-2} = \{2,18,24\}$$

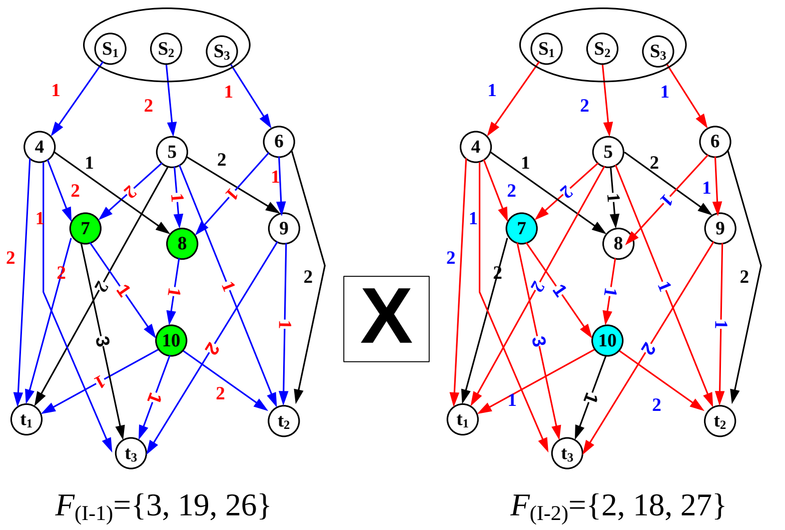
$$\mathit{Parents} \qquad\qquad \mathit{Offspring}$$

$$(a) \qquad\qquad (b)$$

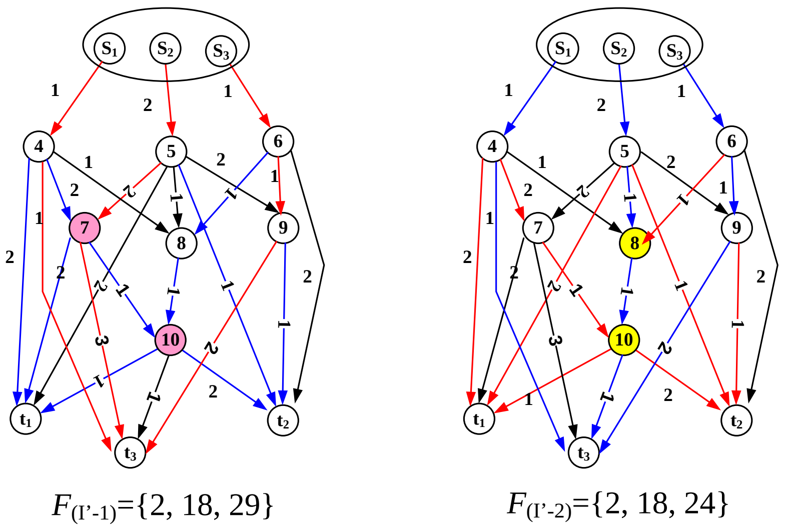
Figure 4-12: Crossover operation at gene level

Assuming the generation (P_t) does not satisfy the termination criteria and the iteration process performs on the generation (P_t) to form the next generation (P_{t+1}). The optimal Pareto (F_{OP}^{t+1}) for the generation (P_{t+1}) is calculated as $F_{OP}^{t+1} = \{2,17,24\}$.

Figure 4-12 (a) shows these two individuals, and their graph level representations are shown by Figure 4-13 (a). These individuals are closer to the optimal Pareto (F_{OP}^{t+1}), therefore they are selected as the parents. The crossover operation works on them to form the offspring population (Q_{t+1}). The offspring are shown in Figure 4-12 (b) as the gene level representation and in Figure 4-13 (b) as the graph level representation. They show fitness increases compared to their parents.



(a) Parents



(b) Offspring

Figure 4-13: Crossover operation at graph level

4.4.2.2.2 Mutation

The mutation operator introduces random changes into chromosome characteristics and is generally applied at the gene level. In typical GA implementations, the mutation rate (probability of changing the properties of a gene) is very small and depends on the length of the chromosome. Therefore, the new chromosome produced by mutation should not be very different from the original one. Mutation plays critical role in GA. The crossover operator leads to population convergence and mutation reintroduces genetic diversity back into the population assisting the search to escape from local optima.

Here, the length of the chromosome is bounded by a number of sinks so if a gene is randomly substituted by mutation, the original chromosome is significantly changed by the resultant high mutation rate of *1/N*. This is addressed by operating on a single path in a randomly selected gene and without perturbing a linear disjoint feature of the gene, thus reducing the mutation rate by a factor of *h*.

$$\begin{array}{l}
\quad \textit{Randomly selected gene} \\
\qquad\qquad \Downarrow \\
\left[\!\left[\left\{\begin{matrix}\langle S_1,4,t_1\rangle \\ \langle\ \ \rangle \\ \langle S_3,6,8,10,t_1\rangle\end{matrix}\right\} \left\{\begin{matrix}\langle S_1,4,7,10,t_2\rangle \\ \langle S_2,5,t_2\rangle \\ \langle S_3,6,9,t_2\rangle\end{matrix}\right\} \left\{\begin{matrix}\langle S_1,4,t_3\rangle \\ \langle S_2,5,7,t_3\rangle \\ \langle S_3,6,9,t_3\rangle\end{matrix}\right\} \right]\!\right] \\
\quad in \Uparrow \qquad \underbrace{\langle S_2,5,7,t_1\rangle}_{\Downarrow} \\
\overbrace{\langle S_2,5,t_1\rangle}^{\Uparrow} \qquad\quad \Downarrow out
\end{array}$$

Figure 4-14: Mutation operation at gene level

The offspring $F_{(I'-1)}$ in Figure 4-13 (b) is formed by the crossover operation and the path of a random gene is mutated. Figure 4-14 shows the mutation operation representation at the gene level and Figure 4-15 shows the same representation at the graph level. The gene of the offspring is randomly selected for the mutation operation, and its path is randomly substituted by another linearly independent path. For a example, the first gene of $F_{(\Gamma'-1)}$ has been selected for the mutation operation and its second path $\langle S_2, 5, 7, t_1 \rangle$ has been randomly substituted by a linearly disjoint path $\langle S_2, 5, t_1 \rangle$. Figure 4-15 (b) shows the offspring after mutation and its fitness remains unchanged. Therefore this example provides evidence that the proposed mutation method prevents the significant divergence of the original offspring or chromosome.

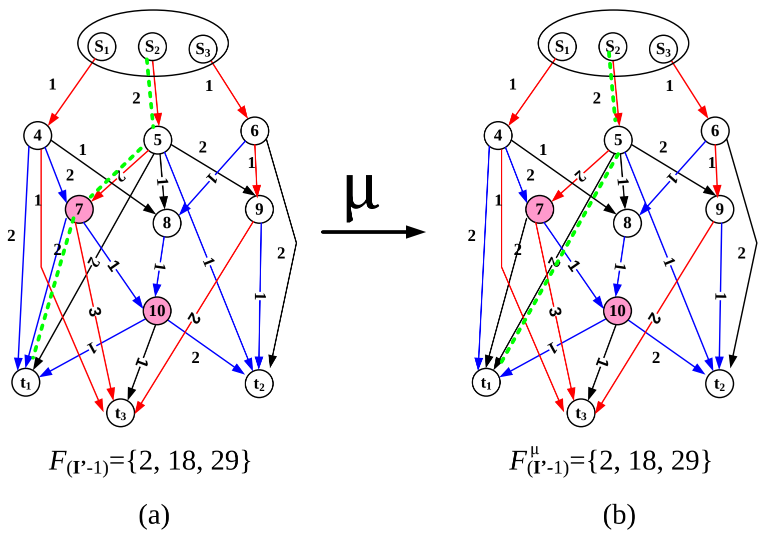

S1
S2
S3
4
5
6
7
8
9
10
t1
t2
t3
μ
$F_{(\Gamma'-1)}$={2, 18, 29}
(a)
$F^{\mu}_{(\Gamma'-1)}$={2, 18, 29}
(b)

Figure 4-15: Mutation operation at graph level

4.4.2.2.3 Selection

A selector operator plays a vital role in this work because it may pull the search to a narrow area of search space. The selector operator is connected with the fitness assignment and individual evaluation, (Section - 4.4.2.1). The selector operator selects K of the offspring in the offspring population Q_t based on their fitness and they are copied into the generation P_{t+1}, where K is the population size.

The selector operator performs differently in MOGA and VEGA, as shown in Figure 4-16. The GA operators of crossover and mutation work on a mating pool to form the offspring population Q_t. The selector operator creates two different mating pools for MOGA and VEGA. The generation P_t are assigned their fitness using the objective functions and they are evaluated using the Pareto optimal F_{t+1}^{OP}. The selector operator in MOGA concerns closer individuals to the Pareto optimal F_{t+1}^{OP} and the MOGA mating pool is filled by them. For example, the individuals I_1, I_4 and I_6 are in Figure 4-16. But the selector operator in VEGA concerns closer individuals to each objective of the Pareto optimal F_{t+1}^{OP} and the VEGA mating pool is filled by them. For examples, the individuals from I_1 to I_6 are closer to $f_{t+1}^{OP}(X)$, the individuals I_1, I_2, I_6 are closer to $f_{t+1}^{OP}(Y)$ and individuals I_4, I_5 are closer to $f_{t+1}^{OP}(Z)$, and the VEGA matting pool is filled by them. Moreover the offspring population Q_t are evaluated using the Pareto optimal F_{t+1}^{OP} and the selector operator performs on Q_t as same as the selector operator on MOGA.

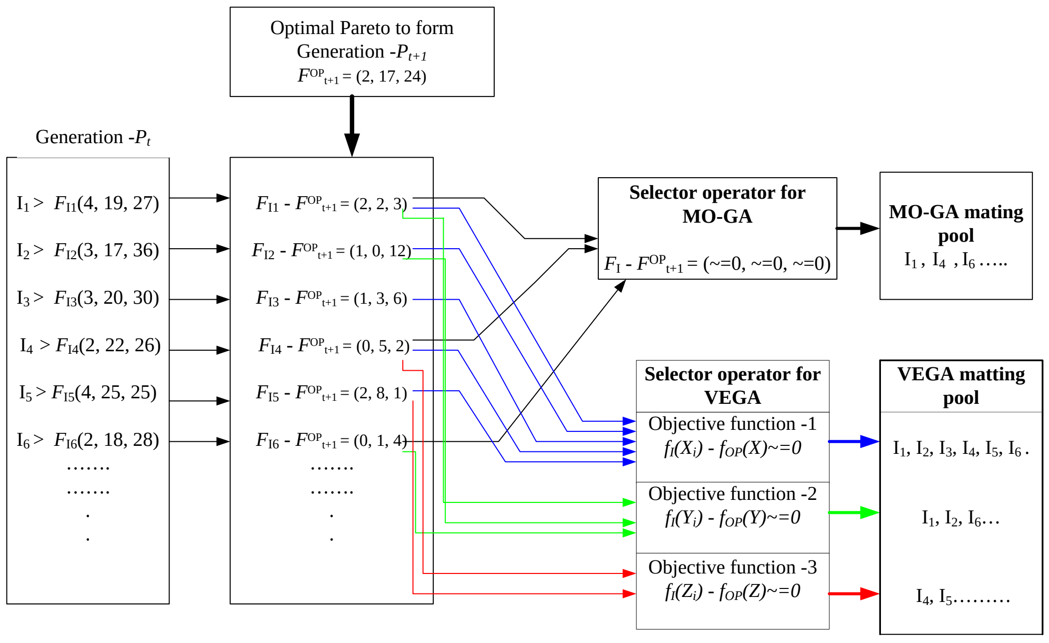

Optimal Pareto to form Generation $-P_{t+1}$ $F^{OP}_{t+1} = (2, 17, 24)$
Generation $-P_t$
$I_1 > F_{I1}(4, 19, 27)$
$I_2 > F_{I2}(3, 17, 36)$
$I_3 > F_{I3}(3, 20, 30)$
$I_4 > F_{I4}(2, 22, 26)$
$I_5 > F_{I5}(4, 25, 25)$
$I_6 > F_{I6}(2, 18, 28)$
$F_{I1} - F^{OP}_{t+1} = (2, 2, 3)$
$F_{I2} - F^{OP}_{t+1} = (1, 0, 12)$
$F_{I3} - F^{OP}_{t+1} = (1, 3, 6)$
$F_{I4} - F^{OP}_{t+1} = (0, 5, 2)$
$F_{I5} - F^{OP}_{t+1} = (2, 8, 1)$
$F_{I6} - F^{OP}_{t+1} = (0, 1, 4)$
Selector operator for MO-GA
$F_I - F^{OP}_{t+1} = (\sim=0, \sim=0, \sim=0)$
MO-GA mating pool
I_1, I_4, I_6.....
Selector operator for VEGA
Objective function -1
$f_I(X_i) - f_{OP}(X) \sim= 0$
Objective function -2
$f_I(Y_i) - f_{OP}(Y) \sim= 0$
Objective function -3
$f_I(Z_i) - f_{OP}(Z) \sim= 0$
VEGA matting pool
$I_1, I_2, I_3, I_4, I_5, I_6$.
I_1, I_2, I_6...
I_4, I_5.........

Figure 4-16: How the selector operator works on MOGA and VEGA

4.4.2.2.4 Termination criteria

A process of chromosome generations is terminated when criterion conditions are met. When the termination criteria are met, the fitter chromosomes are returned as the best solutions found so far.

This investigation focuses on implementing the whole algorithms in the source node, executing them to identify the feasible minimal configurations (individuals). Therefore if the source identifies a number, *w*, being the feasible minimal configurations, then the process is terminated, or else the process continues. Moreover, if the termination condition is not met during *n* generations, the entire population is removed and the process randomly re-initiated.

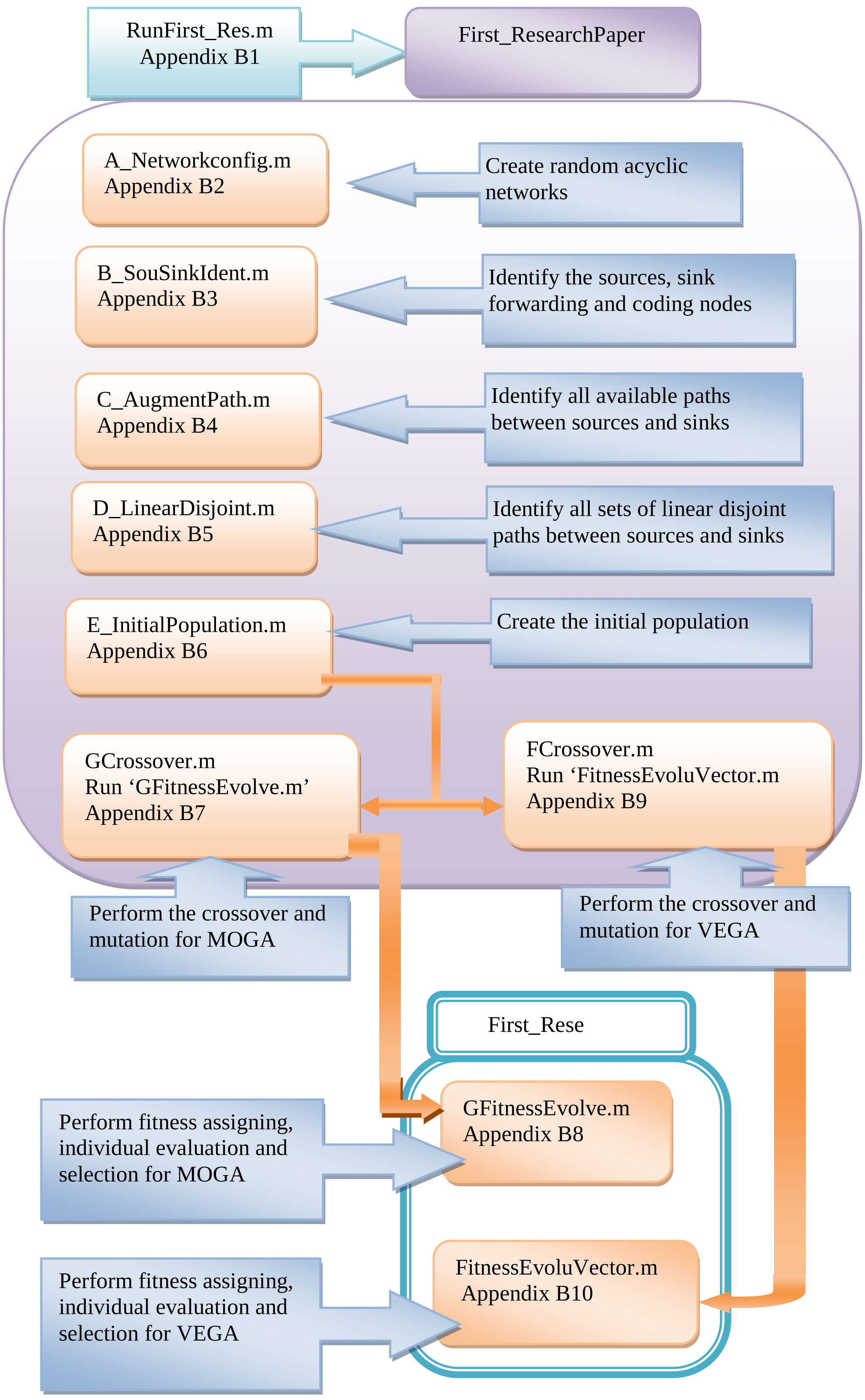

RunFirst_Res.m
Appendix B1
First_ResearchPaper
A_Networkconfig.m
Appendix B2
Create random acyclic networks
B_SouSinkIdent.m
Appendix B3
Identify the sources, sink forwarding and coding nodes
C_AugmentPath.m
Appendix B4
Identify all available paths between sources and sinks
D_LinearDisjoint.m
Appendix B5
Identify all sets of linear disjoint paths between sources and sinks
E_InitialPopulation.m
Appendix B6
Create the initial population
GCrossover.m
Run 'GFitnessEvolve.m'
Appendix B7
FCrossover.m
Run 'FitnessEvoluVector.m
Appendix B9
Perform the crossover and mutation for MOGA
Perform the crossover and mutation for VEGA
First_Rese
GFitnessEvolve.m
Appendix B8
FitnessEvoluVector.m
Appendix B10
Perform fitness assigning, individual evaluation and selection for MOGA
Perform fitness assigning, individual evaluation and selection for VEGA

Figure 4-17: Simulation test bed

4.5 SIMULATION SETUP

To undertake a credible simulation it is essential to have a reliable infrastructure, and this will now be described. Figure 4-17 shows the software implementation of the simulation test bed.

4.5.1 SIMULATION PHASE

Five different randomly generated topologies were used; each consisted of a single source with three data streams, and a different number of nodes, links and sinks.

4.5.1.1 Simulation Parameters

The GA parameters were: Population size (p_z), Crossover probability (pr_c), Mutation probability (pr_μ) and Termination criterion (*w*). They were represented as a parameter set $\{p_z, pr_c, pr_\mu, w\}$. The mutation probability (pr_μ) was decreased as the number of sinks increased. Each run continued until pre-defined generation number (*g*) after which if the GA had not converged to the termination criterion (*w*) then this constituted a ‘failed search’, otherwise the solution was recorded.

4.5.1.2 Results

The tests proceeded as four projects, each of which consisted of a different number of runs. In each run, an equal size topology was employed but it was randomly generated upon commencement. An example is shown in Figure 4-18 for run 1 of project 1 and consists of 27 nodes, 57 links and 07 sinks.

Nodes 1-3 3 symbolize three data streams of the source *S* and Nodes 21-27 symbolize the sinks with all other nodes being forwarding or coding nodes.

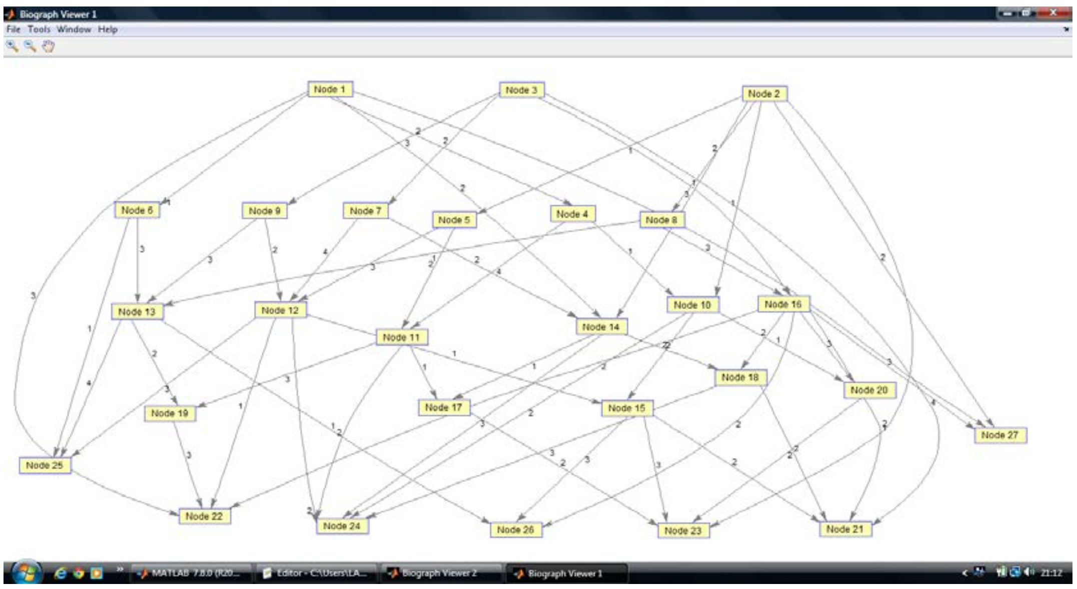


Figure 4-18: A randomly generated topology for run – 1 of project – 1

This section discusses the simulation outcomes of the GA for run 1 of project 1, where the number of sinks was 7, and there were 3 data streams, giving a mutation probability of 0.05. The termination criterion was that at least four feasible individuals were identifiable within ten generations, giving a parameter {100, 0.7, 0.05, 04}.

Figure 4-19 shows the simulation results for the initial population evolution; there is a feasible individual identified by MOGA and VEGA, shown in the sequence Figure 4-19(a)-(f). Figure 4-19(b) shows infeasible individuals in the initial population. Figure 4-19(c), (d) and (e) show, the individuals evaluated by the objective functions $f_I(X_i - X_1^{OP})$, $f_I(Y_j - Y_1^{OP})$ and $f_I(Z_k - Z_1^{OP})$ in order. Individual 33 is qualified by the objective functions which imply that it is more closed to Pareto optimal (F_1^{OP}) and the individual identified by VEGA as the feasible individual. The initial population cannot satisfy the termination criterion and the next generation is created by the GA operations.

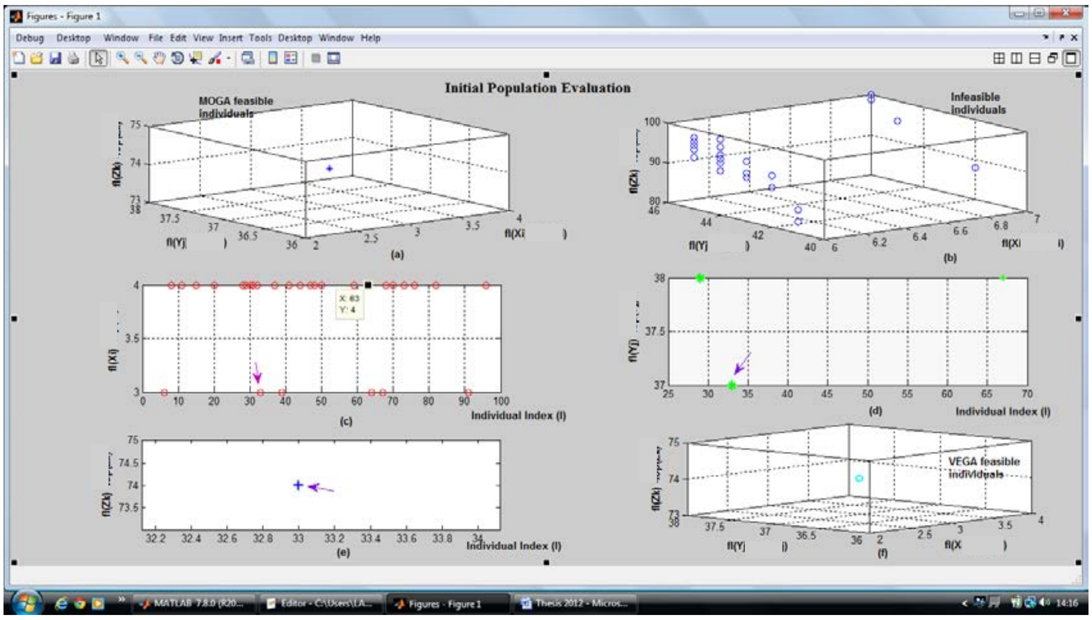


Figure 4-19: Initial population evaluation for run -1 of project - 1

Figure 4-20 shows the simulation results for the third generation evaluation. As shown in Figure 4-20 (a), MOGA and VEGA satisfy the termination criterion with respective CPU times of 131.07 and 107.58 seconds (Table 4-1: Simulation results for project - 1). Figure 4-20 (b) shows infeasible individuals attempting to converge towards the Pareto Optimum. Figure 4-20 (c)-(e) show that more individuals are qualified by the objective functions, during the third generation.

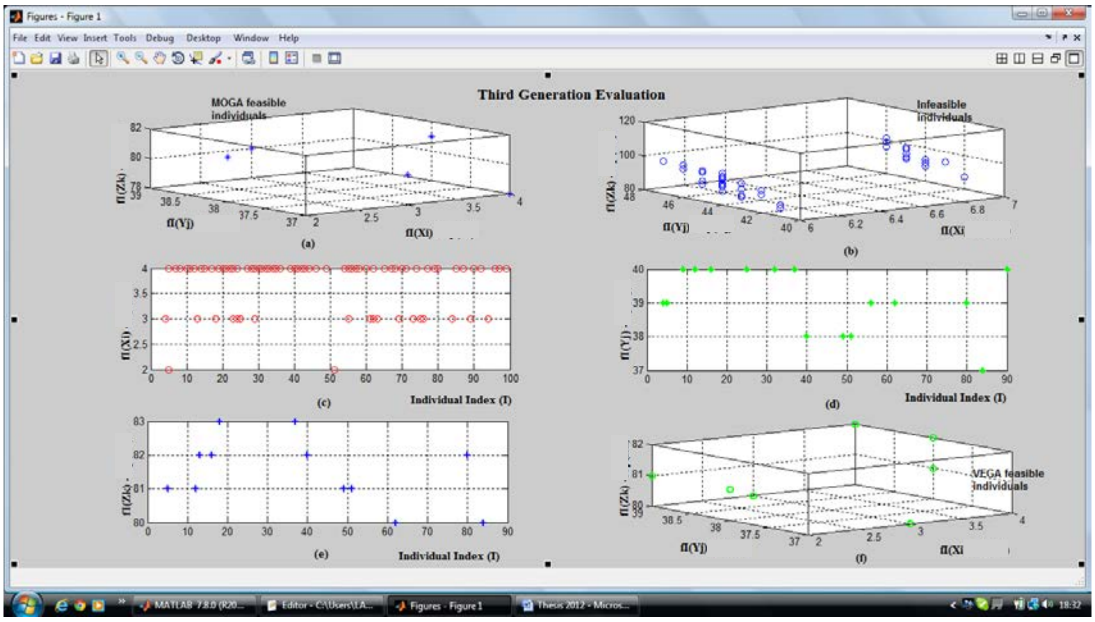
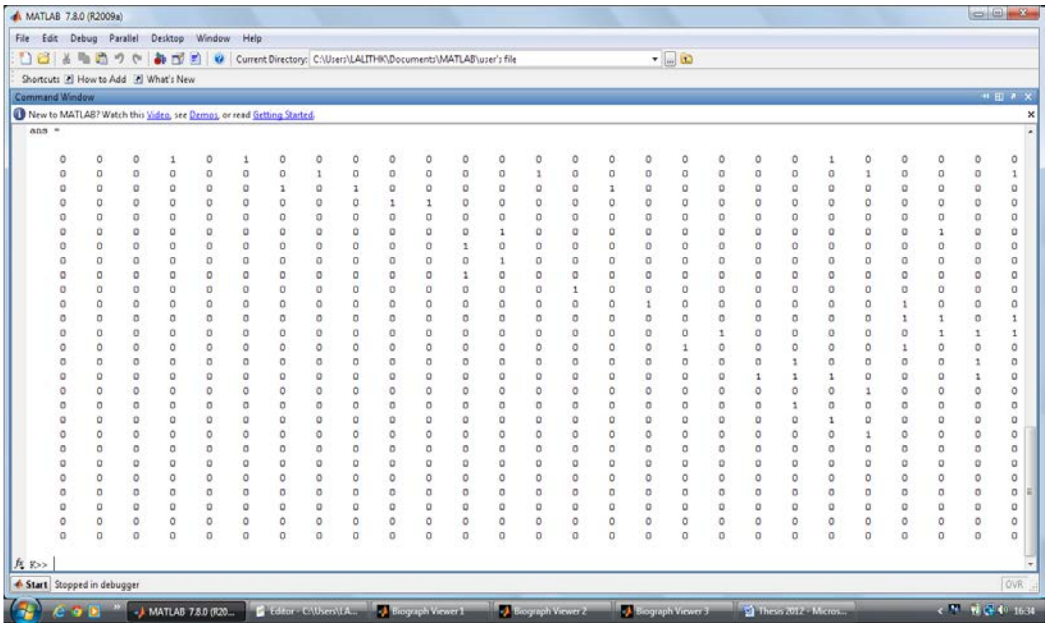


Figure 4-20: Third generation evaluation for run -1 of project - 1

Figure 4-21 shows, an identified sparse matrix of the feasible multicast structure and its graphical representation. Node5 in Figure 4-21(b) does not contribute to any operation during the multicast transmission.



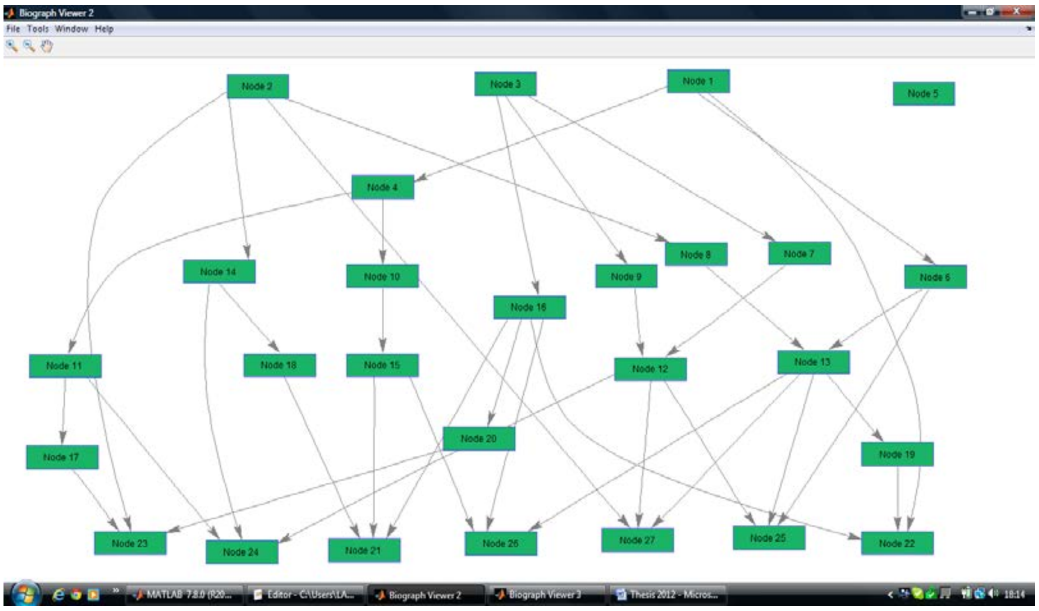
```
ans =

     0     0     0     1     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0
     0     0     0     0     0     0     0     1     0     0     0     0     0     1     0     0     0     0     0     0     0     0     1     0     0     0     1
     0     0     0     0     0     0     1     0     1     0     0     0     0     0     0     1     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     1     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     1     0     1
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     1     1     1
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     1     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     1     1     0     0     0     1     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

K>>
```

MATLAB 7.8.0 (R2009a)
File Edit Debug Parallel Desktop Window Help
Current Directory: C:\Users\LALITHK\Documents\MATLAB\user's file
Shortcuts How to Add What's New
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
Start Stopped in debugger
OVR
MATLAB 7.8.0 (R20...
Editor - C:\Users\LA...
Biograph Viewer 1
Biograph Viewer 2
Biograph Viewer 3
Thesis 2012 - Micros...
16:34

(a)

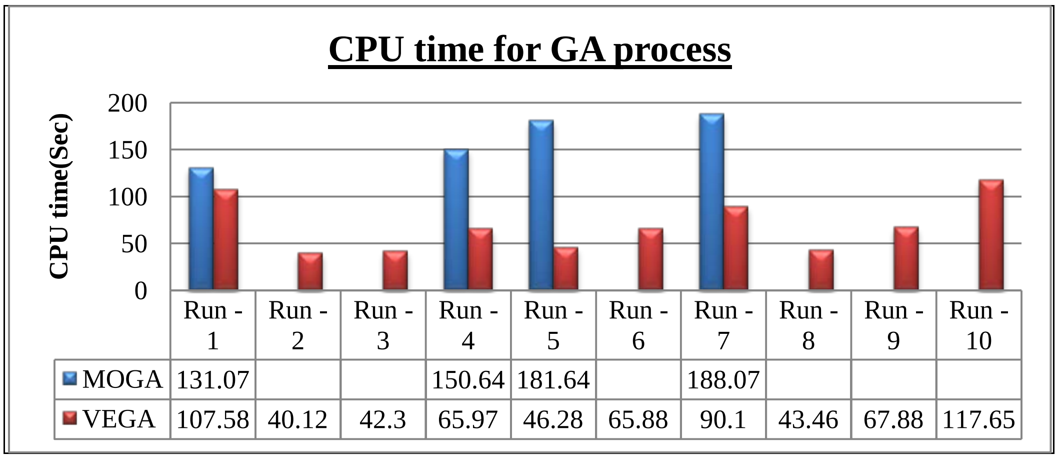


(b)

Figure 4-21: (a) A sparse matrix for an identified minimal configuration in run -1 of project – 1 and (b) its graphical representation

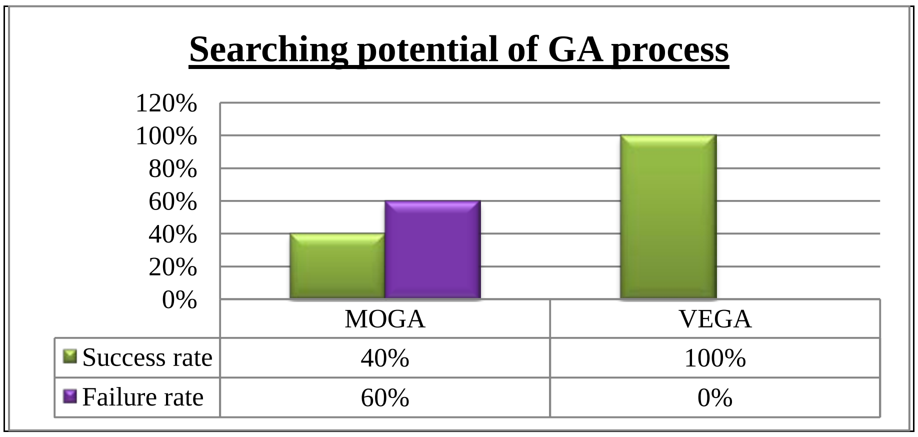
Run	Topological detail			CPU time for Preliminary process (Second)		CPU time for GA Process(Sec){100,0.7, pr_{μ} ,4}	
	Nodes	Links	Sinks	Augmenting Paths Algorithm	Linear Disjoint Paths Algorithm	MOGA	VEGA
1	27	57	07	0.35	3.14	131.07	107.58
2				0.24	2.35	Failed	40.12
3				0.24	2.30	Failed	42.30
4				0.21	2.33	150.64	65.97
5				0.35	2.85	181.64	46.28
6				0.29	2.12	Failed	65.88
7				0.28	2.37	188.07	90.10
8				0.26	2.49	Failed	67.88
9				0.33	2.31	Failed	43.47
10				0.23	2.39	Failed	117.65

Table 4-1: Simulation results for project - 1



	Run - 1	Run - 2	Run - 3	Run - 4	Run - 5	Run - 6	Run - 7	Run - 8	Run - 9	Run - 10
🟦 MOGA	131.07			150.64	181.64		188.07			
🟥 VEGA	107.58	40.12	42.3	65.97	46.28	65.88	90.1	43.46	67.88	117.65

(a)



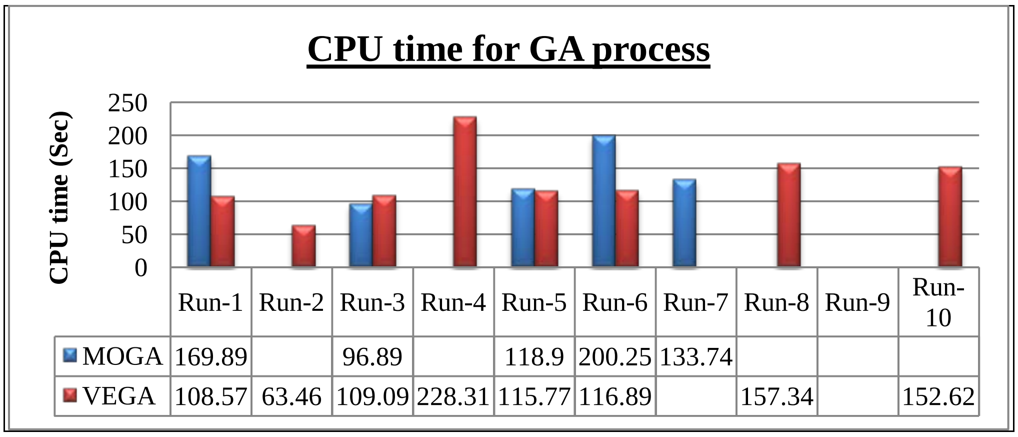
	MOGA	VEGA
🟩 Success rate	40%	100%
🟪 Failure rate	60%	0%

(b)

Figure 4-22: Simulation results analysis for project – 1

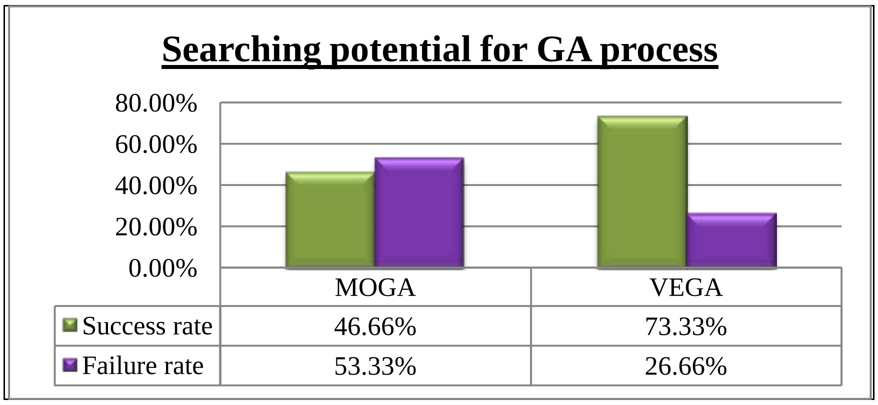
Run	Topological detail			CPU time for Preliminary process (Second)		CPU time for GA Process(Sec){100,0.7 pr_μ ,4}	
	Nodes	Links	Sinks	Augmenting Paths Algorithm	Linear Disjoint Paths Algorithm	MOGA	VEGA
1	30	68	07	0.37	4.28	169.89	108.57
2				0.44	3.77	Failed	63.46
3				0.28	3.60	96.89	109.09
4				0.34	4.44	Failed	228.31
5				0.30	4.65	118.90	115.77
6				0.41	4.87	200.25	116.89
7				0.45	3.53	133.74	Failed
8				0.27	3.65	Failed	157.34
9				0.27	3.67	Failed	Failed
10				0.60	5.33	Failed	152.62

Table 4-2: Simulation results for project – 2



	Run-1	Run-2	Run-3	Run-4	Run-5	Run-6	Run-7	Run-8	Run-9	Run-10
🟦 MOGA	169.89		96.89		118.9	200.25	133.74			
🟥 VEGA	108.57	63.46	109.09	228.31	115.77	116.89		157.34		152.62

(a)



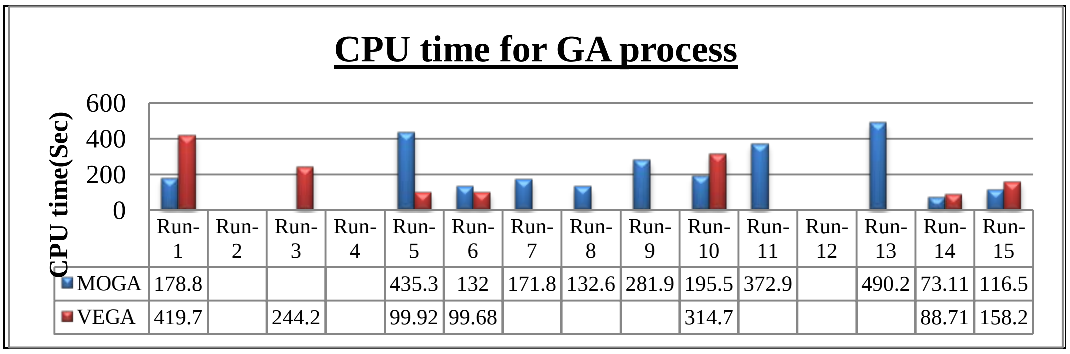
	MOGA	VEGA
🟩 Success rate	46.66%	73.33%
🟪 Failure rate	53.33%	26.66%

(b)

Figure 4-23: Simulation results analysis for project – 2

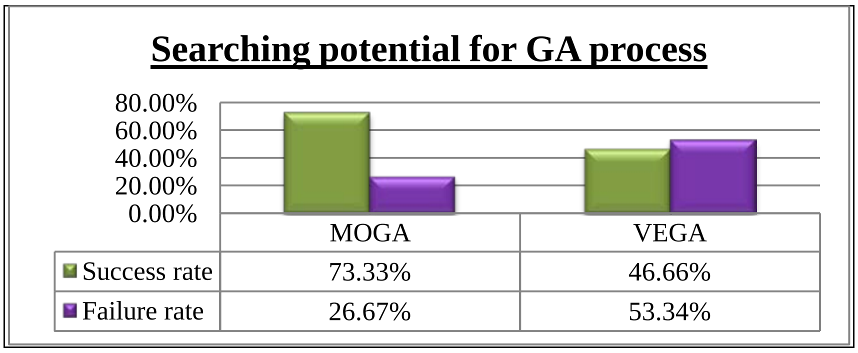
Run	Topological detail			CPU time for Preliminary process (Second)		CPU time for GA Process (sec) {100,0.7, pr_{μ} ,4}	
	Nodes	Links	Sinks	Augmenting Paths Algorithm	Linear Disjoint Paths Algorithm	MOGA	VEGA
1				0.54	18.52	178.76	419.73
2				0.60	20.96	Failed	Failed
3				0.60	17.29	Failed	244.23
4				0.69	17.54	Failed	Failed
5				0.64	17.85	435.26	99.92
6				0.66	18.11	131.97	99.68
7				0.54	18.64	171.83	Failed
8	35	92	12	0.76	20.99	132.62	Failed
9				0.72	27.26	281.90	Failed
10				0.94	25.77	195.45	314.67
11				1.11	33.11	372.90	Failed
12				1.11	30.33	Failed	Failed
13				0.81	28.07	490.24	Failed
14				0.69	24.59	73.11	88.71
15				1.21	29.58	116.48	158.20

Table 4-3: Simulation results for project – 3



	Run-1	Run-2	Run-3	Run-4	Run-5	Run-6	Run-7	Run-8	Run-9	Run-10	Run-11	Run-12	Run-13	Run-14	Run-15
🟦 MOGA	178.8				435.3	132	171.8	132.6	281.9	195.5	372.9		490.2	73.11	116.5
🟥 VEGA	419.7		244.2		99.92	99.68				314.7				88.71	158.2

(a)



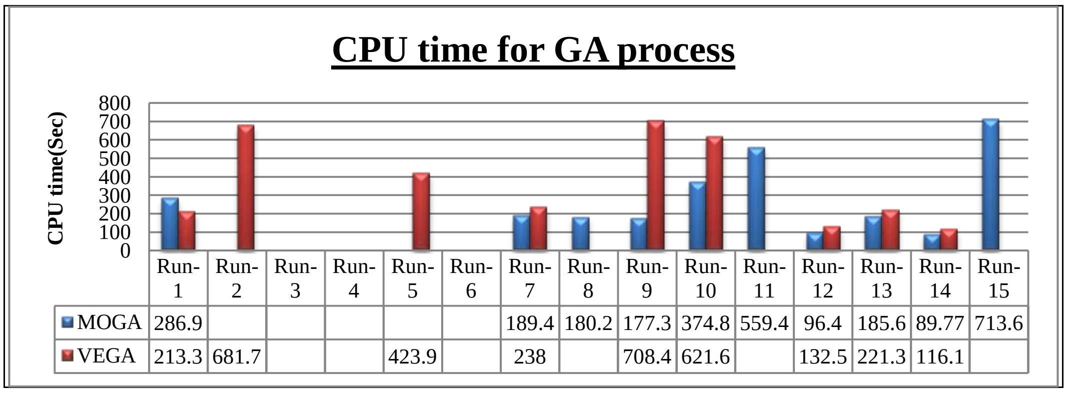
	MOGA	VEGA
🟩 Success rate	73.33%	46.66%
🟪 Failure rate	26.67%	53.34%

(b)

Figure 4-24: Simulation results analysis for project – 3

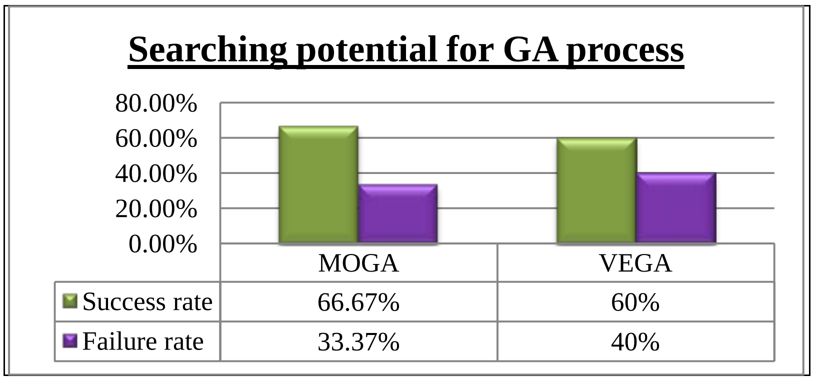
Run	Topological detail			CPU time for Preliminary process (Second)		CPU time for GA Process (Second){100,0.7, pr_μ ,4}	
	Nodes	Links	Sinks	Augmenting Paths Algorithm	Linear Disjoint Paths Algorithm	MOGA	VEGA
1				0.82	29.60	286.85	213.31
2				1.44	42.17	Failed	681.68
3				1.01	40.65	Failed	Failed
4				0.80	43.95	Failed	Failed
5				1.05	42.45	Failed	423.94
6				1.31	44.44	Failed	Failed
7				1.64	41.76	189.37	238.01
8	40	113	17	1.76	47.51	180.22	Failed
9				0.86	40.88	177.25	708.40
10				1.37	39.98	374.84	621.62
11				2.21	47.94	559.45	Failed
12				1.31	48.53	96.40	132.50
13				0.93	42.61	185.63	221.30
14				0.98	41.96	89.77	116.06
15				1.45	42.55	713.62	Failed

Table 4-4: Simulation results for project – 4



	Run-1	Run-2	Run-3	Run-4	Run-5	Run-6	Run-7	Run-8	Run-9	Run-10	Run-11	Run-12	Run-13	Run-14	Run-15
🟦 MOGA	286.9						189.4	180.2	177.3	374.8	559.4	96.4	185.6	89.77	713.6
🟥 VEGA	213.3	681.7			423.9		238		708.4	621.6		132.5	221.3	116.1	

(a)



	MOGA	VEGA
🟩 Success rate	66.67%	60%
🟪 Failure rate	33.37%	40%

(b)

Figure 4-25: Simulation results analysis for project – 4

4.6 RESULTS AND DISCUSSION

For each project a number of simulations were executed. Each project used a different size topology and consisted of a different number of runs, each of which employed a randomly-generated topology. The results demonstrate the potential to identify the minimal configurations between the source and sinks, and they are comprised the optimised network and coding resources. For example, Figure 4-21 (b) shows the identified minimal configuration whose identification is actually shown in Figure 4-20 (f). It consists of two coding nodes (Node 12 and Node 13), thirty eight links, and eighty links' distances.

Close inspection of Figure 4-21 (b) shows that it obeys the minimum cut capacity -maximum flow theorem. When *S* feeds three different data streams into Nodes 1-3, and either these streams are coded or not by intermediate nodes, all sinks are able to obtain simultaneously the multicast data via the sets of the linear disjoint paths.

These simulations do not attempt to deliver the actual multicast traffic levels, rather they identify the minimal source to sink configurations, which is NP-hard. The performance of the proposed solution is considered in two parts, the preliminary process and the evolutionary process. Figure 4-26 shows the performance of the two preliminary algorithms as a function of increasing scale (project) analysis for the simulations in the all projects. The path augmentation is largely independent of network size, in contrast to the linear disjoint path algorithm, which has a more difficult task to perform as the network gets larger.

With respect to the evolutionary process, the search performance of the two multi-objective GA techniques MOGA and VEGA differed as the network size varied. The two algorithms were applied to the same initial population for testing. For project 1, Figure 4-22 (a) shows the CPU time for MOGA and VEGA, and the latter exhibits superior performance. Moreover, Figure 4-22 (b) shows that VEGA was also superior in its searching as it is did not fail to find a solution unlike MOGA. Project 2 used networks and scales that are much higher than project 1. As Figure 4-23 (a) shows, MOGA had a slight improvement on the CPU time and VEGA a slight degradation. Figure 4-23 (b) shows that the searching potential of MOGA improved by 6.66% by comparing with project 1, but VEGA shows a corresponding significant degradation. Nevertheless, VEGA still showed good performance over MOGA in project 2. Moving to projects 3 and 4, where the networks were large, the algorithm performances are shown in Figure 4-24 and Figure 4-25. VEGA showed good performance rather than MOGA, in terms of both on the CPU time and the searching potential for small scale networks, but MOGA performance in both ways got better as the network scale was increased. This is a significant observation based on the simulation results, VEGA is ideal for searching small scale networks and MOGA is good for searching large scale networks. An exact reason was not clear this observation, but just the decision is vital, based on the simulation results.

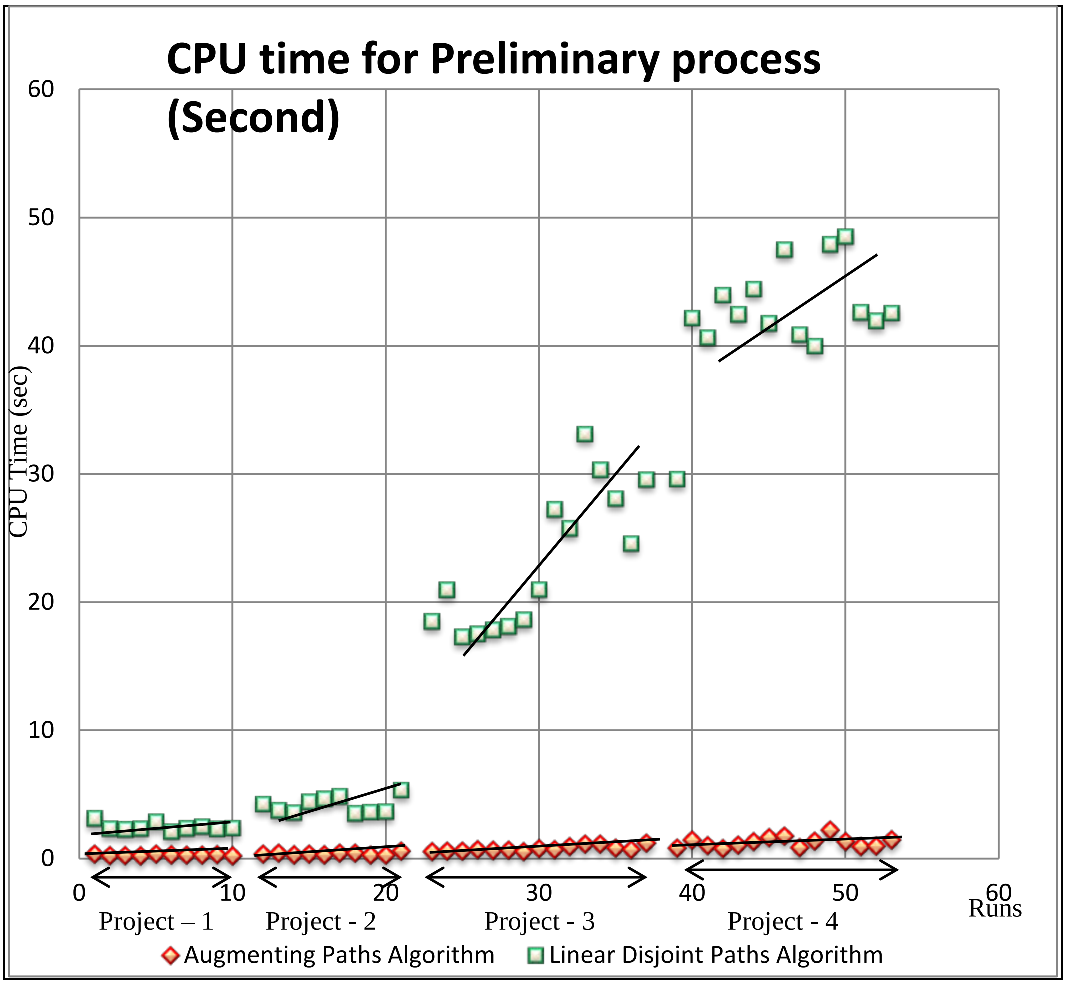


Figure 4-26: Preliminary process analysis for all projects

4.7 CONCLUSION

This part of the research is a first exploration of an evolutionary algorithm based on GA approaches to network code design in the multicast scenario. The research activities provide new pathways for researchers in the field to expand the network coding concept in the multicast scenario. Among the number of network coding problems, the evolutionary approach is necessary to solve those which are categorised as NP-hard. Therefore, the problem is defined as the identification of the minimal configuration between the source and the sinks. In addition, the complexity of the problem is increased, because the minimal configuration identified should comprise the optimum network and coding resources.

The complexity of network codes construction and network coding protocol development is tacked via the identification of the minimal configuration. The solution here contributes to minimise complexity problems, and does not require dramatic alterations of a well-established network infrastructure. Moreover, functional integrations of the network nodes are not necessary to execute the algorithms. The algorithmic solution is designed to be implemented in the source nodes, since these are enriched with high computational resources, such as memory and processing capabilities. The solution allows the intermediate nodes to perform their fundamental operations, such as forwarding and coding only. Therefore the solution contributes to an escape from a costly functional integration of the intermediate nodes.

Simulation results from the augmenting path algorithm as preliminary process showed good performance in terms of the CPU time, and this was largely independent of network size. This was in marked contrast to the linear disjoint path algorithm which has a more difficult task to perform as the network gets larger. Moreover, in the simulation results of the evolutionary process, VEGA showed good performance rather than MOGA, in terms of both the CPU time and the searching potential for small scale networks but MOGA performance in both ways improved as the network scale was increased.

4.8 REFERENCES

[1] C. Fragouli and E. Soljanin, “Network Coding Fundamentals”, *Foundation and Trends in Networking, Vol 2*, no.1, pp.1-133,2007.

[2] Donald E. Knuth. The art of computer programming, volume 2 (3rd ed.): Seminumerical algorithms. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[3] Takuji Nishimura and Makoto Matsumoto. Mersenne twister home page.

[4] M. B. Richey and R. G. Parker, “On multiple steiner subgraph problems,”*Networks*, vol. 16, no. 4, pp. 423–438, 1986.

[5] M. Langberg, A. Sprintson, and J. Bruck, “The encoding complexity of network coding,” IEEE Trans. Inf. Theory, vol. 52, no. 6, pp. 2386–2397, 2006.

[6] Holland JH. Adaptation in natural and artificial systems. Ann Arbor:University of Michigan Press; 1975.

[7] Goldberg DE. Genetic algorithms in search, optimization, and machine learning. Reading, MA: Addison-Wesley; 1989.

[8] Zitzler E, Deb K, Thiele L. Comparison of multiobjective evolutionary algorithms: empirical results. Evol Comput 2000;8(2):173–95.

[9] Schaffer JD. Multiple objective optimization with vector evaluated genetic algorithms. In: Proceedings of the international conference on genetic algorithm and their applications, 1985.

[10] Fonseca CM, Fleming PJ Multiobjective genetic algorithms. In: IEE colloquium on ‘Genetic Algorithms for Control Systems Engineering’ (Digest No. 1993/130), 28 May 1993. London, UK: IEE; 1993.

[11] Y. and Fabregat R. Donoso, *Multi-Objective Optimization in Computer Networks Using Metaheuristics*. Boca Raton, FL, U.S.A.: Auerbach Publications, 2007.

5 EVOLUTIONARY APPROACH FOR NETWORK CODING RESOURCE OPTIMISATION

Before recent advances in multicast coding techniques, multicast transmission was extremely demanding and consumed considerable network resources such as channel bandwidth and network power. Hence, research efforts have focused on minimising network resource usage, introducing novel and efficient network coding techniques [1]. Prior to this, network nodes only performed packet routing, forwarding and duplicating functions. The novel multicast coding technique of network coding employed an additional function at the intermediate nodes of the network to combine two of more independent bits streams via binary addition or linear combination. At present, considerable efforts are being made to minimise the coding resources in the multicast scenario [1], [2]. In this chapter, the evolutionary approach is proposed to solve the problem.

5.1 THE PROBLEM AND ITS CONTEXT

Here the communication network is represented by a directed acyclic graph $G = (V, E)$ with unit capacity edges and that the value of the min cut between the source and each of the sinks is h. The source node *S* is required to transmit simultaneously, *h*, unit-rate independent information streams $\{s_1, s_2 ...... s_h\}$, and a set of *N* sinks $\{t_1, t_2 ..... t_N\}$ is required to receive the multicast data from the source *S*. The source needs to apply the multicast coding technique in this multicast transmission and it requires identification of the minimal configurations between the source itself and the set of *N* sinks. Moreover these minimal configurations have abilities to minimise the network coding resources during the multicast transmission.

Fundamentally, the coding nodes are enriched in terms of buffer memory, computational capability and operating power, and these additional abilities are defined as the coding resources. These resources are rapidly consumed and ultimately exhausted by computational complexity, packet delay, congestion, packet misrouting and so forth. The packet delay, congestion and packet misrouting contribute to cause synchronising errors at the coding nodes and decoding errors at the sinks.

The network coding resources for multicasting are comprehensively discussed by Fragouli and Soljanin [1] who describe the major complexity components as *Set-up complexity* and *Operational complexity.* The former denotes the complexity of designing the network coding scheme, which includes selecting the paths through the information flows and determining the operations (coding, forwarding etc.) that the nodes of the network perform. The latter encompasses the running cost of using network coding, that is, the amount of computational and network resources required per information unit successfully delivered. Moreover, this complexity is strongly correlated with the network coding scheme employed. For example, Figure 4-3 shows the coding scheme which can be used to deliver the multicast traffic with optimum network and coding resources usages.

The operational complexity is further discussed using assumptions that the source S simultaneously emits multicast packets $\{\sigma_1, \sigma_2 ......\sigma_h\}$ which are elements of some finite field $\mathbb{F}_q$, and they are transmitted via the minimal configuration G' of G

consisting of hN paths. In linear network coding, these elements are linearly combined and forwarded by some intermediated nodes of *G'*, and these combined packets are elements of $\mathbb{F}_q$. The linear combination of h information streams requires $\mathrm{O}(h^2)$ finite field operations. The complexity is further affected by the *size of the finite field* over which operations take place as the cost of finite field arithmetic grows with the field size. For example, typical algorithms for multiplication or inversion over a field of size $q = 2^n$ require (n^2) binary operations. Moreover the field size affects the required storage capabilities at intermediate network nodes. The computational complexity is further affected by the *number of coding points* in G'. Coding points are, in general, more expensive due to need to equip them with encoding capabilities. In addition, coding points incur delay and increase the overall complexity of the network [3]. The computational complexity at each coding point of *G'* is considerably increased by a number of *in-links* per coding point and which exhausts the coding resources via increasing operational network complexity [3].

To recover the source packets $\{\sigma_1, \sigma_2 ......\sigma_h\}$, which have been linearly combined over $\mathbb{F}_q$ by the coding nodes, each sink needs to solve a system of $h \times h$ linear equations, which requires $\mathrm{O}(h^3)$ operations over $\mathbb{F}_q$ if Gaussian elimination is used.

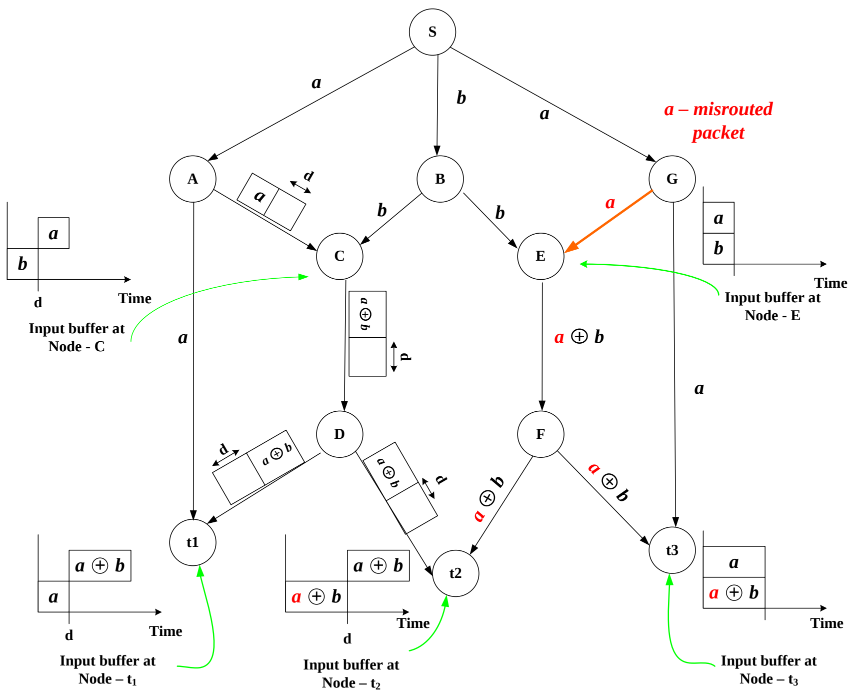


Figure 5-1: Congestion, packets delay and packet misrouting exhaust network coding resources and cause decoding errors

Figure 5-1 is used to explain the issues mentioned in section 5.1 and their effects on the network coding resources. Packet ‘*a*’ is congested in link *AC* by a packet, in fact it is delayed by time *d*. Node *C* is receiving packet ‘*b*’ on link *BC* and it has to store this in a Node C input buffer during the time *d* until ‘*a*’ arrives. This is defined as a ‘*synchronous error to coding operation*’ and consumes Node C’s power to maintain its buffer memory. As a result, coded packet $'a \oplus b'$ is routed throughout the network with time delay *d*. Therefore Sinks t_1 and t_2 face a synchronous error like Node C and this is defined as a ‘*synchronous error to decoding operation*’. Moreover Node *G* misroutes packet ‘*a*’ through link *GE* and an unwanted coding operation proceeds at Node E. The coded packet $'a \oplus b'$ is routed throughout the network and t_2 is able to receive identical packets $'a \oplus b'$and ‘$a \oplus b'$ meaning it is impossible for this sink to obtain the original packets ‘*a*’ and ‘*b*’ by solving linear equations. This issue is defined as a ‘*decoding error*’ and t_2 re-requests the multicast data from *S*, which then attempts to redeliver the multicast data ‘*a*’ and ‘*b*’ not only to t_2 but also to t_1 and t_3. Therefore network and coding resources are allocated to retransmit the same set of the multicast data ‘*a*’ and ‘*b*’. The decoding error causes fatal damage to the network and coding resources.

5.2 WORKS RELATED TO THE PROPOSED SOLUTION

The proposed solution intends to identify the minimal configurations between the source and the set of sinks and these configurations have the capability to save coding resources during multicast transmission. The solution uses evolutionary algorithms based on GAs.

This problem is somewhat similar to that of the “Travelling Salesman Problem (TSP)” [4] and in both cases GAs may be employed to search for the suitable geometrical properties. The TSP is the classic NP-hard problem in combinatorial optimisation studies and an optimal solution for even moderate size problems is intractable. Given a list of cities and their pair wise distances, the task is to find the shortest possible route that visits each city exactly once and returns to the original city. The TSP is modelled as an undirected weighted graph, such that cities are the graph’s vertices, paths are the graph’s edges, and a path’s distance is the edge’s length. Considering the complexity of NP-hard problems, a GA is employed to solve the problems efficiently.

Unlike source coding, network coding is performed by a lower layer device such as a router with the capability of mixing its inputs. Therefore this kind of router has special capabilities, such as mathematical manipulation, buffer memory maintenance and operational power management. Comprising these capabilities, the traditional router is converted to an expensive piece of equipment and it is of natural interest to reduce the number of such devices deployed, whilst satisfying the communication demand. The problem of determining a minimal set of nodes where coding is required to achieve the given multicast rate is NP-hard, because this decision can be taken by reducing the problem into a multiple Steiner subgraph problem, which is NP-hard [5]. As Figure 5-2 shows, the butterfly network is reduced to the multiple Steiner subgraph problem to take the decision on the minimal set of the coding nodes, which are required to achieve the multicast rate 2.

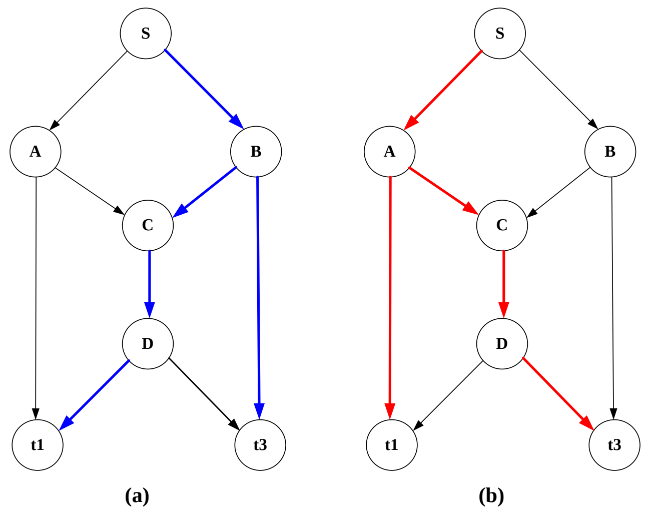


Figure 5-2: Multiple Steiner subgraphs

Fragouli *et al.*[6] show that coding is required at no more than $(d-1)$ nodes in acyclic networks, with two unit-rate sources and d sinks. The butterfly network in Figure 5-2 (a) contains two unit-rate sources and the two sinks, and so requires coding at only one node. The disadvantage of this result is that it cannot be generalised to more than 2 sources. Moreover [6] introduced an algorithm to construct a minimal subtree graph that has been discussed previously in Section 3.2.4. To achieve the target rate R, the algorithm initially selects a subgraph consisting of R link-disjoint paths to each of d sinks. The given network is transformed to a labelled line graph and each link is sequentially examined and removed if its removal does not affect the achievable rate.

Langberg *et al.* [3] derive an upper bound on the number of required coding nodes for both acyclic and cyclic networks. The bounds depend only on the desired multicast rate and the number of sinks. In this method, the given network is transformed into a new network in which each node has at most degree three. The new network is used to obtain a minimal subgraph by sequentially examining and removing the edge whether its removal does not affect to the achievable rate. The bounds are calculated for the obtained minimal subgraph. Moreover it is also shown in [3] that approximating the minimum number of coding points is NP-hard.

Kim *et al.* [2] investigate two algorithms which were proposed in [6] and [3]. Each algorithm removes the edges of the given network to find a suboptimal solution (a minimal sub-graph), in a greedy fashion[5] and assumes all intermediate nodes of the remaining graphs can perform the network coding with their incoming links. Figure 5-3 shows how these approaches are able to lead to the suboptimal solution in a simple network. Assume that edge l in Figure 5-3(a) has capacity 2, which it is represented in Figure 5-3(b) as two parallel unit-capacity links l_1 and l_2. Here it may also be mentioned that the additional capacity allows the achievement of a multicast rate of 2 without network coding. In Fragouli *et al.*'s approach, either link l_1 or link l_2 may be removed while selecting the subgraph, network coding is necessary at

[5] A greedy algorithm is performed in a greedy fashion and it follows the problem solving heuristic of making the locally optimal choice at each stage with the expectation of searching a global optimum.

node C to achieve the multicast rate 2. Moreover whether coding is required depends on the order in which the links are visited to construct a minimal subtree graph; for example, if the order of link inspection is randomly chosen, then coding is required with probability 0.5.

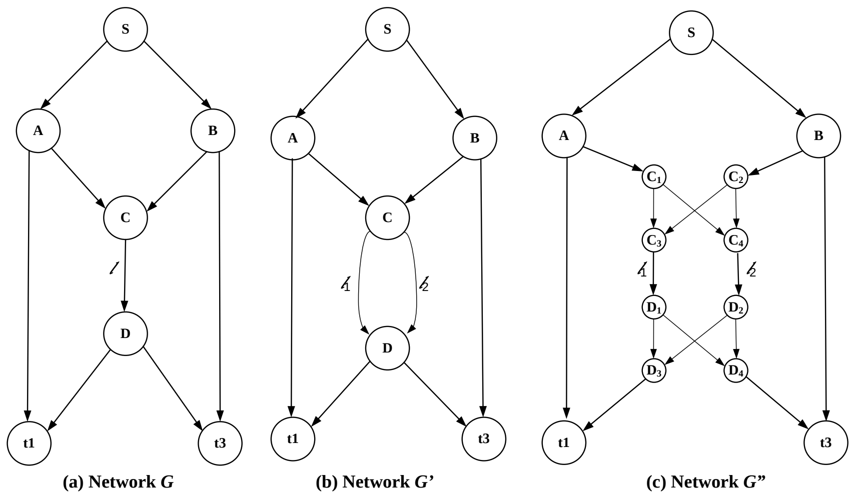


Figure 5-3: Sample networks

Langberg *et al.*'s approach initially decomposes nodes C and D as in Figure 5-3(c). This network consists of many sequences of link removals that result in a subgraph where coding is required, for example, if l_1 is the first visited link then node C_4 must perform coding. An Empirical test shows that, if the order of link inspection is randomly chosen, then coding is required with probability 0.68.

Kim *et al.* [2] observe in the above two approaches that finding a good order of link transversal in a large number of many possible sequences may be detrimental to the quality of solutions. These two approaches do not contain any method to evaluate the solutions obtained. Therefore, they may cause the problem that the decision as to where to perform coding involves a selection out of a large number of choices. Figure 5-3 also illustrates a possible trade-off between network coding and link usage. Fragouli *et al.*'s method increases coding in the remaining subgraph when reducing link usage as in the subgraph selection; minimising coding first may increase links usage. Therefore an optimal choice depends on the relative cost of each of the resources; the proposed method in [2] focuses on optimising these two costs.

Kim *et al.* [2] consider the problem of minimising the resources used for network coding while achieving the desired throughput in the multicast scenario. Rather than tackling this NP-hard problem they focus on quickly finding a sufficiently good solution. Their method consists of an evolutionary algorithm based on the GA, with the latter working in an algebraic framework, combined with randomised polynomial identity testing methods.

In Kim *et al.*'s approach, the network is considered to be as the acyclic directed multigraph $G = (V, E)$, where each link has a unit capacity. To represent links with larger capacities, multiple links are allowed between a pair of nodes. Only integer flows are allowed, so there is no flow or a unit rate of flow on each link. The packet transmission is a single source multicast in which the source, $S \in V$, transmits data at rate R to a set of $T \subset V$ sink nodes, where $|T| = d$. The rate, R,is said to be achievable if the transmission scheme available is able to transmit multicast data to all d sinks simultaneously. Given an achievable rate, R, their solution needs to determine a minimal set of nodes where coding is required in order to achieve this rate. The maximum achievable multicast rate is the minimum of the individual max-flow bounds between the source and each of the sinks [8]. Linear network coding is sufficient for multicast [7], and Kim *et al.*'s approach considers that a node's output on the outgoing link is the linear combination of the inputs from its incoming links.

The general network coding problem is algebraically formulated by Koetter *et al* [9], and Kim *et al.*'s approach considers how this algebraic formulation can be applied to the case where network coding is performed only at some subset of the nodes.

In Kim *et al.*'s approach, they first construct the labelled line graph $G' = (V', E')$ corresponding to G [8]. Then each link of G' is assigned a link coefficient, denoted by $\xi_i \in \underline{\xi}$, and system vector is constructed which is necessary to form a system matrix at each of d sinks. Each system matrix is an $R\times R$ matrix which describes how the individual source packets are linearly combined at the intermediate nodes. If all system matrixes of d sinks are full rank over the ring of polynomial in $\underline{\xi}$ [9], then the approach verifies that the given multicast rate R is achievable. But this verification procedure is complicated when several nodes are considered together. Whether coding is required or not at a node depends on whether coding is performed at other nodes; therefore the verification procedure cannot be applied separately to each node. For example, in Figure 1-1(b) with three sinks and a desired multicast rate of two, when Node *C* or Node *E* is tested separately, they show that neither must be a coding node. When all nodes are considered together, coding is required at least at one of Node *C* and Node *E* to achieve the multicast rate of two. When the number of involved nodes is augmented, exponentially large selections of link coefficients are necessarily evaluated to identify where coding may be required. The proposed solution in [2] intends to minimise the number of coding links and the solution is comprised the structure of the standard GA introduced by Holland [10].

Later work by Kim *et al.* [11] is a GA- based solution to minimise the resources used for network coding, and it is a significant improvement of their previous approach [2]. The previous algorithm can be applied only to acyclic network but their new approach [11] is suitable for both acyclic and cyclic networks. The new approach enriches the set of components used in the GA, which improves the performance. Moreover they introduce a novel distributed framework which is combined with the distributed random network coding scheme [12] and the resources used for coding are optimised in the setup phase by running the evolutionary algorithm based on GA at each node of the network.

The major drawbacks of Kim *et al.*'s approaches are:

1. A node where coding is required cannot be decided independently which implies that whether coding is required at a node or not depends on whether coding is performed at other nodes; therefore the verification procedure cannot be applied separately to each node. So, when the number of involved nodes is augmented, the complexity of the verification procedure becomes complicated. For example, the network in Figure 1-1(b) with three sinks and a desired multicast rate of two, when either Node *C* or Node *E* is tested separately, shows that neither must be a coding node. When all nodes are considered together, coding is required at least one of Node *C* and Node *E* for achieve the multicast rate of two.

2. The approach comprises an evolutionary algorithm based on a GA, and the GA operations are performed at each node on an individual basis. It is well known that the GA is a demanding and memory-hungry algorithm, therefore it may cause packet delays, packet misrouting, synchronous errors at the coding nodes and excessive transmission complexities. Most of the

network nodes are located beyond the human access limit such as satellite nodes, under sea nodes, etc., therefore these nodes cannot be integrated on their function, hardware or software; moreover it is a costly process. A large number of nodes in the network perform fundamental operations such as packet forwarding and mathematical operations (binary addition, subtraction etc), and they do not consist of adequate buffer memories and processing capabilities. Considering these drawbacks, the GA operations cannot be efficiently performed in each node. Consequently their approach for optimising the network coding resources contributes to exhaust excessively the operational resources of the network nodes and introduce large scale complexities to the data transmission.

3. A fitness evaluation process of the approach is very inefficient (or even impossible) when the network size is augmented exponentially. Each link in the labelled line graph *G'=(V',E')* is assigned a link coefficient (ξ_i), and a system matrix is constructed for each of *d* sinks, where *R* is the multicast rate. The product of the determinant of those *d* matrices is denoted by $P(\underline{\xi})$. The components of vector $\underline{\xi}$ consist of all link coefficients ξ_i and the link coefficient ξ_i, is selected as the *m*-dimensional binary vector from a finite field $\mathbb{F}_{2^m}$. Each chromosome is represented by *m*-dimensional binary vector, its associated k^{th} link coefficient is ξ_k. If a chromosome $\underline{y}$ is given, then the polynomial $P(\underline{\xi})$ is evaluated to find $\underline{y}$'s feasibility. Each transfer (decoding) matrix $M_i \, (1 \le i \le d)$, which is defined as $M_i = A(I-F)^{-1} B_i^T$ in

[9], has a size $R \times R$ for multicast rate R, and each of its elements is a polynomial consisting of $O(\mu|E|^2)$ terms, where μ is the maximum number of ways to traverse from any link to another in the network, which, in general, grows exponentially with the size of the network. The determinant of M_i thus contains $O(\mu|E|^2)^R.R!$ terms, which makes keeping $P(\underline{\xi})$ in polynomials form very inefficient (or even impossible) for its exponential size.

5.3 PROPOSED SOLUTION TO OPTIMISE NETWORK CODING RESOURCES

The identification of the minimal configuration with optimised network coding resources is NP-hard. The proposed solution, based on a GA, accepts the challenge of solving this and can to quickly identify a solution instead of tackling the NP-hard problem. Section 5.3.1 discusses how the identification of the minimal configuration solves the problems discussed in section 5.1, and the major drawbacks of Kim's approach. Section 5.3.2 explains the proposed solution and its framework.

5.3.1 IDENTIFICATION OF THE MINIMAL CONFIGURATION AND ITS BENEFITS

The two major complexities below conspire to exhaust the network coding resources [1], and there now follows a discussion of how these complexities are optimised by identifying the minimal configuration:-

1. Optimisation of the operational complexity
2. Optimisation of the setup complexity
3. Overcoming the major drawbacks of Kim's approach

The minimal configuration in Figure 5-4(a) has three coding points {'7', '8', '9'}. The two linear disjoint paths $\langle S_1,4,7,t_3\rangle,\langle S_2,5,7,t_1\rangle$ pass through node 7 as its input links, and node 7's input buffer is allocated to store two bits or two packets. Moreover node 7 performs a binary operation $(a\oplus b)$,where $a,b\in\mathbb{F}_2$ [6]. But node 8 is unlike node 7 because it is passed through by three linear disjoint paths $\langle S_1>4>8>10>t_2\rangle,\langle S_2>5>8>10>t_3\rangle$ and $\langle S_3>6>8>10>t_1\rangle$ and its input buffer is allocated to store three bits or three packets. Furthermore node 8 performs a binary operation $(a\oplus b\oplus c)$, where $a,b,c\in\mathbb{F}_2$. Therefore node 8 consumes more coding resources rather than node 7 and node 9.

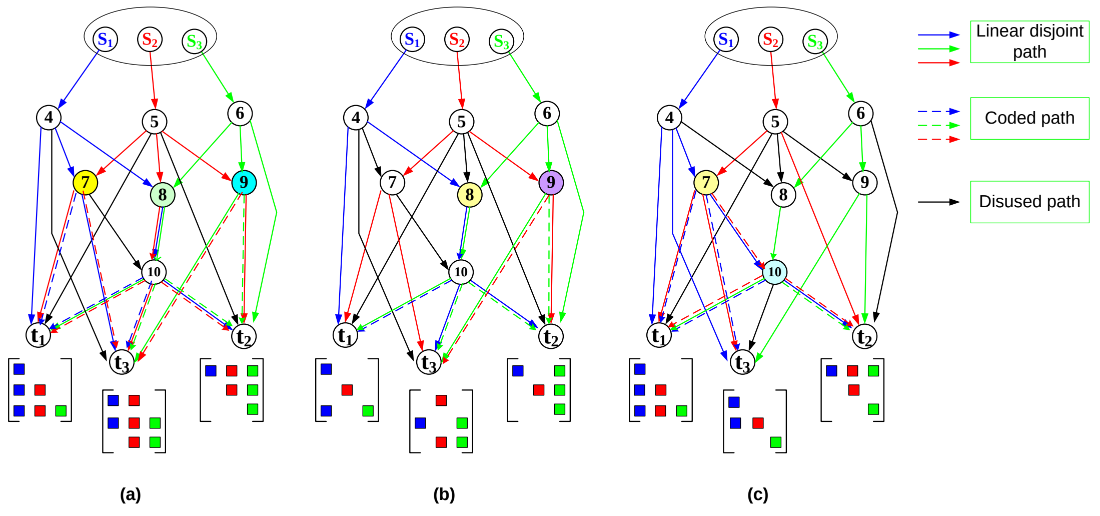


Figure 5-4: The minimal configurations with network coding resources usage; all sinks in each configuration can be simultaneously obtained the full rank matrixes

Considering the network coding resource usage, Figure 5-4(b) and (c) are in a same condition and they are better than Figure 5-4(a). Among the minimal

[6] $\mathbb{F}_2$ is a finite field with two elements {0,1}.

configurations in Figure 5-4 , that in Figure 5-4(c) shows the best performance because it contributes to save more network resources. The numbers of disused paths in Figure 5-4(a), (b) and (c) are 4, 5 and 6, respectively. It is well known that the concept of network coding was introduced to save network resources (link's capacity) during multicast transmission, therefore Figure 5-4(c) is a best selection of the configurations in Figure 5-4. It is interesting how Figure 5-4(c) has become the best selection because it consists of an optimum number of coding nodes (coding resources) and they are optimally shared by all sinks. The coding resources of node 7 are directly shared by sink t_1 and t_3 via links (7>t_1) and (7>t_3), and indirectly shared by sink t_1 and t_2 via paths $\langle 7 > 10 > t_1 \rangle$ and $\langle 7 > 10 > t_2 \rangle$. Moreover the coding resources of node 10 are shared by sink t_1 and t_2 via links (10>t_1) and (10>t_2). In Figure 5-4(c), the coding resource sharing ratio per coding node is 3 (six shared links/a number of coding nodes = 6/2). This ratio is calculated for Figure 5-4(a) and (b) as 2.34 and 2.5 respectively.

Each minimal configuration in Figure 5-4 either contributes to save network coding resources or not, nodes of each minimal configuration with their operations (coding, forwarding etc), and their interconnected paths can be clearly defined by identifying the minimal configuration. The complexities are always built when the paths via nodes and their operations are clearly identified; it is called the setup complexity. It is essential when the network coding scheme is constructed. For example section - 4.2 discusses how the minimal configuration is deployed to construct the network coding scheme.

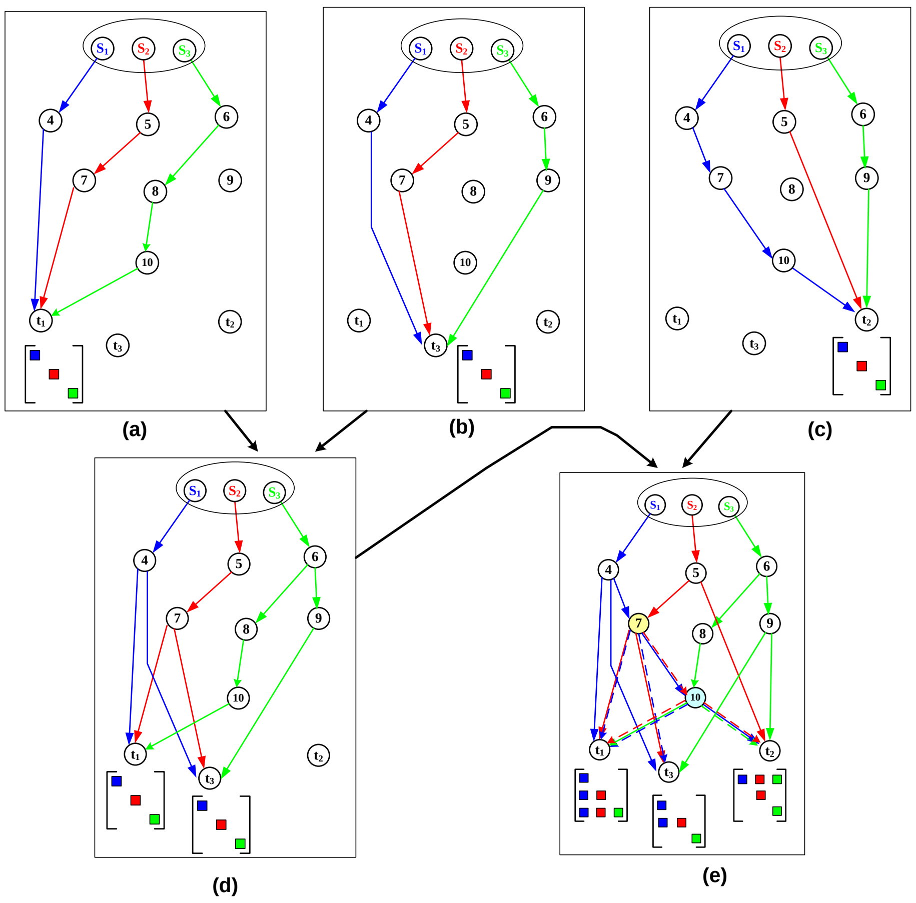


Figure 5-5: (a), (b) and (c) show sets of linear disjoint paths for sink – t_1, t_3 and t_2; (d) shows how the minimal configuration makes for sink – t_1, t_3; (e) shows how the minimal configuration makes for sink – t_1, t_3 and t_2

The solution phase previously shown in Figure 4-7 is applicable to solve the problem formulated in Section-5.1, but the fitness evaluation process is the only difference in comparing the implementation of section 4.4.2.1. In Section-4.4, the preliminary process identifies the different sets of linear disjoint paths from the source to each receiver. The GA process (see Section-4.4.2) combines these sets to form the minimal configuration, and the objective functions in section-5.3.2.1 contribute to constrain the coding resources of which the minimal configuration is composed.

Figure 5-5 (a), (b) and (c) show the sets of the linear disjoint paths for sinks t_1,t_2 and t_3. It is very clear that each sink is entitled to receive the multicast data without errors, because each sink can form an identity matrix. Figure 5-5 (d) shows a combination of the two sets of linear disjoint paths (Figure 5-5 (a) and (b)), and it is a possible minimal configuration for sinks t_1 and t_3. The minimal configuration in Figure 5-5 (d) does not consist of overlapped paths or nodes, which implies that the coding resources are not engaged to its multicast transmission. Moreover, sinks t_1 and t_3 are able to form identity matrixes, and they can obtain simultaneously multicast data without errors. Figure 5-5 (e) shows a combination of the three sets of linear disjoint paths (Figure 5-5 (a), (b) and (c)) and it is a possible minimal configuration for sinks t_1, t_3 and t_2. The minimal configuration in Figure 5-5 (e) consists only of overlapped nodes (node 7 and node 10), which implies that the coding resources are engaged in its multicast transmission. Moreover, sinks t_1, t_3 and t_2 are able to form the full rank matrixes, and they obtain multicast data without errors.

The proposed solution is able to overcome the three major drawbacks of Kim's approach. A node where coding is required can be decided independently which implies that whether coding is required at a node does not depend on whether coding occurs at other nodes. The fitness evaluation process contributes to include the optimum number of coding points to the feasible minimal configuration during the GA operations. The GA operations are not concerned where the coding is required and the minimal configuration independently includes the optimum coding points by the GA operations. Therefore, when the number of involved nodes is augmented (such as a large scale network), the verification procedure does not become complicated.

The intermediate nodes in the network are unlike the source node. The source consists of adequate memory and processing capacity, and these resources are essential to perform the GA operations. The proposed solution is thus implemented at the source node. The GA operations find the feasible minimal configurations for the source to deliver its multicast traffic. Hence, the source node is the only one required to have hardware and software modifications. The intermediate nodes are only allowed to perform their fundamental operations (packet routing and coding). Therefore the proposed solution is significant in overcoming the second major drawback of Kim’s approach.

The fitness evaluation process of the proposed solution focuses on optimising the coding resources only, and it does not concern all sinks which can form full rank matrices. As in Figure 5-5, and, based on the explanation above, whether coding resources are optimised or not, all sinks of the minimal configuration can form full rank matrices without any complexities. For example, Figure 5-4 shows the three different minimal configurations which comprise different quantities of coding resources, but all sinks in each configuration can form full rank matrices. Therefore, the proposed approach contributes to prevent the third major drawback of Kim’s approach.

5.3.2 THE PROPOSED SOLUTION AND ITS FRAMEWORK

The entire process of the proposed solution has been thoroughly discussed in section 4.4, and its framework shown in Figure 4-7. However, the fitness evaluation process here differs from the process of section 4.4.2.1. Therefore, this section is only concerned with the fitness evaluation process, simulation results, and their discussion.

5.3.2.1 Fitness Assignment and Individual Evaluation $F_I = \{f_I(X_i), f_I(Y_j), f_I(Z_k)\}$

The fitness evolution process of the proposed solution concentrates on optimising the network coding resources in the multicast scenario. The proposed solution is intended to identify the minimal configurations between the source and sinks, and the fitness evaluation process contributes to optimising the coding resources in those minimal configurations. The fitness evaluation process consists of three objective functions. The individuals in the initial population or mating pool are assigned their fitness following the objective functions. These three objective functions are addressed below to optimise the network coding resources.

1. Optimise the number of coding nodes in individuals - $f_I(X_i)$;
2. Achieve a desired throughput rate (constraining a number of in-links at each coding point) - $f_I(Y_j)$;
3. Optimally share coding resources in individuals - $f_I(Z_k)$.

The first two objectives optimise the network coding resources; the first and third optimise network resources. If an optimum number of coding nodes are in the multicast routes of the minimal configuration, then the multicast transmission consumes the optimum coding resources when that minimal configuration is selected by the source for its multicast transmission. The use of coding nodes in multicast transmission automatically implies that a number of channels convey simultaneously more than one packet, contributing to efficient channel capacity use and network resource savings.

The second objective allows the source to maintain a desired throughput rate during its multicast transmission. This can be achieved by constraining the number of input links at each coding point. Moreover, it allows the saving of coding resources (storage capacity and computation) at the coding nodes.

The third objective allows the sharing of the optimum coding resources with all sinks, and may be endorsed by considering the average coding resource sharing per coding node, defined as the sum of the number of receivers connected to each coding node divided by the number of coding nodes. In addition, it also improves the usage of the coding resources that are discovered via the first two objectives.

The problem is thus one of multi-objective optimisation, and such cases generally exhibit conflicting objectives, preventing the simultaneous optimisation of each. In this case, the first and third objectives are in direct conflict, since, when the number of coding points is optimised, they are unlikely to be evenly spread. Here, the standard GA is customised to accommodate multi-objective problems by using specialised fitness functions, and introducing methods to promote solution diversity. The approach is to determine an entire Pareto optimal solution set rather than a single fitness calculation in traditional GA. It is a most suitable solution, because neither the first nor third objectives have pre-identified constraints. Therefore, the Pareto optimal solution is updated at each generation by comparing it with the one obtained in the previous generation.

It is assumed that the source intends to identify w minimal configurations. Minimal configuration I may be viewed as a point in the solution space $\{f_I(X_i), f_I(Y_j), f_I(Z_k)\}$. The points ($X_{OP}$, Y_{OP}, Z_{OP}) are objective constraints and the feasible set of them forms a Pareto optimal front. Figure 5-6 (a) shows the objective constraints and Figure 5-6 (b) shows a surface that is Pareto optimal on Z_{OP}. The Pareto optimal surface is updated at each generation with the first arising from the randomly selected initial population. The value of Y_{OP} is maintained to be ≥2 but X_{OP} and Z_{OP} are updated at each generation with the minimum value being preferred for X_{OP} and the maximum value for Z_{OP}. For example, Figure 5-6 (b) shows Pareto optimal (OP_{t-1}, OP_t and OP_{t+1}) for generation-$(t-1), t, (t+1)$ consecutively and they are updated at each generation. Pareto optimal (OP_{t-1}) is: $(X_{OP}^{t-1}, Y_{OP}^{t-1}, Z_{OP}^{t-1})$ and OP_t is: $(X_{PO}^{t}, Y_{PO}^{t}, Z_{PO}^{t})$. The comparison of ($OP_{t-1}$) and OP_t is: $([X_{PO}^{t-1} > X_{PO}^{t}], [Y_{PO}^{t-1} > Y_{PO}^{t}], [Z_{PO}^{t-1} = Z_{PO}^{t} = Z_1])$. Therefore the Pareto optimal (OP_{t-1}) is moved to the position (OP_t) on surface Z_1.

The mutual comparison between individuals is extremely challenging in multi-objective optimisation and the proposed method can avoid the difficulty of comparison. At each selection operation, the individuals are assigned their fitness $\{f_I(X_i), f_I(Y_j), f_I(Z_k)\}$ using objective functions. Then each individual is compared with the Pareto optimal (X_{PO}, Y_{PO}, Z_{PO}), using $f_C[(X_i - X_{OP}) \geq 0, (Y_j - Y_{OP}) \geq 0, (Z_k - Z_{OP}) \geq 0]$. If any individual is far away from the Pareto optimal, it can be defined as a weakly fitter or infeasible individual. With reference to Figure 5-6 (b), the individual I_3 on surface Z_1 is in this position but individual I_1 is a fitter individual that should be selected in preference.

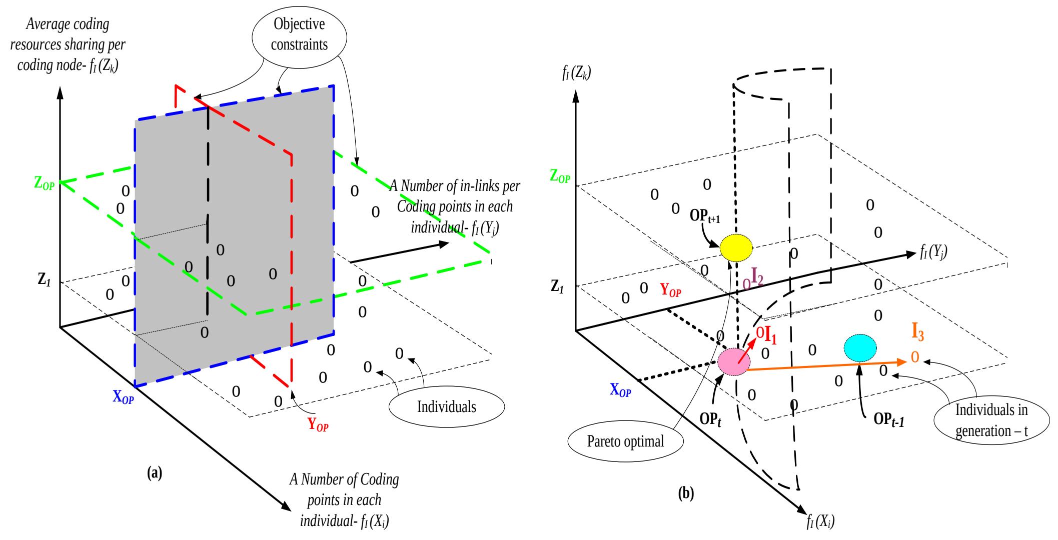

Average coding resources sharing per coding node- $f_I(Z_k)$
Objective constraints
Z_{OP}
Z_I
X_{OP}
Y_{OP}
A Number of in-links per Coding points in each individual- $f_I(Y_j)$
Individuals
A Number of Coding points in each individual- $f_I(X_i)$
(a)
$f_I(Z_k)$
OP_{t+1}
I_2
I_1
I_3
$f_I(Y_j)$
OP_t
OP_{t-1}
Pareto optimal
Individuals in generation – t
$f_I(X_i)$
(b)

Figure 5-6: Pareto optimisation process for the problem considered (a) objective constraints; (b) Pareto optimal front.

As shown in Figure 4-8 and Figure 4-9, individuals are in a path-based format which is hard to analyse at the fitness assignment process stage. Therefore each individual is converted to a sparse matrix as shown in Figure 5-7, where {7, 8 and 10} can be identified as coding nodes because {7} is connected to {4} and {5}, which are in turn connected to sources {S_1} and {S_2}. Moreover {8} is connected to {S_2} and {S_3} via {5} and {6}. Node {10} is connected to coding nodes {7} and {8}. Then objective function $f_I(X_i)$ can be calculated as 3.

The '1's entries of all coding nodes are counted and objective function $f_I(Y_j)$ can be calculated by taking a value of average '1' entries per coding point (6/3).

The objective function $f_I(Z_k)$ calculation process is: sort all the sinks' columns for entries '1'. If entry '1' is found in the coding node's row, then it is counted. Also if a first coding node is connected to a second coding node, and the second coding node connects to a sink, then the sink is contributed by two coding points. In Figure 5-7 (b), sink {t_1} has entry '1' at row – 7 (coding node 7), sinks {t_1, t_2, t_3} have entries ‘1’s at row – 10 (coding node 10) and also node {10} is connected to coding node {7} and {8}. Therefore sinks {t_1, t_2, t_3} are contributed by coding nodes {7}, {8} and {10}. Consequently the objective function $f_I(Z_k)$ can be calculated for Figure 5-7 (b) as (10/3), and the individual’s fitness is: f(3,2,10/3).

$$\left[\!\left[\begin{Bmatrix} \langle S_1, 4, t_1 \rangle \\ \langle S_2, 5, 7, t_1 \rangle \\ \langle S_3, 6, 8, 10, t_1 \rangle \end{Bmatrix} \begin{Bmatrix} \langle S_1, 4, 7, 10, t_2 \rangle \\ \langle S_2, 5, t_2 \rangle \\ \langle S_3, 6, 9, t_2 \rangle \end{Bmatrix} \begin{Bmatrix} \langle S_1, 4, t_3 \rangle \\ \langle S_2, 5, 8, 10, t_3 \rangle \\ \langle S_3, 6, 9, t_3 \rangle \end{Bmatrix} \right]\!\right]$$

(a)

$$\begin{array}{c|ccccccccccccc}
 & S_1 & S_2 & S_3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & t_1 & t_2 & t_3 \\
\hline
S_1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_3 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

(b)

Figure 5-7: (a) An individual in a path format; (b) A related sparse matrix

5.3.2.2 Simulation Results and Discussion

Table 5-1 shows the results of initial simulations of the methods described above for randomly generated networks with the parameters (nodes, links, sinks) shown. The GA parameter set was {100, 0.7, 0.06, 5} and the simulations were run for ten generations using three data streams from the source. Several features are apparent from these results. The available data paths between the source and sinks were rapidly identified by the path-augmenting algorithm. Moreover, the identification of the sets of linear disjoint paths by the linear disjoint path algorithm is also relatively fast. The MOGA and VEGA stages consume considerably more time, as would be expected, given that they are performing a stochastic search of a very large workspace. MOGA offers advantages in general over VEGA, in that it is generally faster, and a little more likely to satisfy the termination criteria during the ten generations.

Topological details			CPU time for Preliminary process (Seconds)		CPU time for GA Process (Seconds) {100,0.7, 0.06, 5}	
Nodes	Links	Sinks	Augmenting Paths Algorithm	Linear Disjoint Paths Algorithm	MOGA	VEGA
25	51	5	0.21	1.65	14.7	18.3
26	54	5	0.25	1.77	90.0	131.3
26	54	6	0.29	1.93	55.4	62.3
26	54	6	0.19	2.37	98.5	89.5
25	54	6	0.20	2.33	fails	fails
26	58	6	0.24	2.63	114.3	fails

Table 5-1: Initial Simulation Results

Table 5-2 shows simulation results for five randomly-generated topologies using the same parameters as above, but for population sizes of 500 and 1000, in addition to the 100 already employed. The details of the random topologies were (Nodes, Links, Sinks): RT1 (25, 54, 6); RT2 (25, 51, 5); RT3 (26, 58, 6); RT4 (26, 57, 6); RT5 (26, 59, 5). For each topology, ten runs were performed, and the table shows the mean CPU times used and the mean number of failures (*F*) for each of the random topologies. It may be observed that VEGA generally takes considerably longer than MOGA and produces marginally more failures. Thus, MOGA is to be preferred as the selection algorithm for this process.

	Average CPU time in seconds for GA Process											
Topology	K = 100				K = 500				K = 1000			
	MOGA	F	VEGA	F	MOGA	F	VEGA	F	MOGA	F	VEGA	F
RT 1	66.5	1	79.3	1	104.3	0	131.6	0	198.7	0	241.4	0
RT 2	129.4	0	141.5	0	185.3	1	352.2	0	315.2	1	387.6	0
RT 3	78.7	0	101.6	1	157.2	0	254.4	2	245.5	0	311.8	1
RT 4	59.2	0	104.4	0	179.2	0	189.2	1	211.3	0	287.5	2
RT 5	81.4	0	79.3	0	211.3	1	332.8	0	256.5	1	389.5	1

Table 5-2: Simulation Results

5.4 CONCLUSION

There are many situations where multicast is required and it has historically presented a demanding challenge in terms of network resources such as channel bandwidth and network power. The introduction of network coding offers the prospect of substantial reductions in resource requirements and this has been addressed here. The solution presented consists of a preliminary process and a GA optimisation stage. The former deals with the aspects of path augmentation and linear disjoint path determination and produces a set of possible multicast structures to deliver traffic from the source to multiple sinks. These consist of three features (objectives) and which are optimised simultaneously during the multicast transmission. Searching for the optimum choices of paths is NP-hard, so heuristic methods (MOGA and VEGA) are employed. Simulations show that MOGA is better than VEGA at efficiently identifying feasible multicast structures. Moreover, it also returns a lower search failure rate. The approach taken has shown itself to be of great utility in minimizing complexity and resource demands, laying the foundations for efficient multicast network schemes for future traffic delivery.

5.5 REFERENCES

[1] C. Fragouli and E. Soljanin, “Network Coding Fundamentals”, *Foundation and Trends in Networking, Vol 2*, no.1, pp.1-133,2007.

[2] M. Kim, C. W. Ahn, M. Médard, and M. Effros, “On minimizing network coding resources: An evolutionary approach,” Network Coding Workshop, 2006.

[3] M. Langberg, A. Sprintson and J. Bruck, “The Encoding Complexity Of Network Coding”, *IEEE Trans. Information Theory*, 2006, vol.52, no.6, pp.2386 - 2397.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, “Introduction to Algorithms”, 2nd edtion. MIT Press, Cambridge Massachusetts, 2001.

[5] M. B. Richey and R. G. Parker, “On multiple steiner subgraph problems,” *Networks*, vol. 16, no. 4, pp. 423–438, 1986.

[6] C. Fragouli and E. Soljanin, “Information flow decomposition for network coding,” *IEEE Trans. Inform. Theory*, to appear.

[7] S.-Y. R. Li, R. W. Yeung, and N. Cai, “Linear network coding,” *IEEE Trans. Inform. Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[8] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, “Network information flow,” *IEEE Trans. Inform. Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[9] R. Koetter and M. M´edard, “An algebraic approach to network coding,” *IEEE/ACM Trans. Networking*, vol. 11, no. 5, pp. 782–795, 2003.

[10] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press, 1975.

[11] M. Kim, V. Aggarwal, U.-M. O’Reilly, M. M´edard, and W. Kim, “Evolutionary approach to minimising network coding resources,” in Proc. IEEE INFOCOM’07, May 2007.

[12] T. Ho, R. Koetter, M. M´edard, D. R. Karger, and M. Effros,“The benefits of coding over routing in a randomized setting,”in Proc. IEEE ISIT, 2003.

6 EVOLUTIONARY APPROACH FOR SECURE NETWORK CODING

The joint optimisation of network parameters in the multicast scenario is a complex process and the evolutionary approach appears essential to overcome its complexity. Among the many network parameters, in which the network user community is interested, the two most vital parameters of both network security and cost are essentially optimised in a correlated manner. These two parameters have been jointly investigated by Tan *et al.* [1] as a first attempt. This chapter discusses how these two parameters are optimised simultaneously for multicast transmission. The first parameter, network security for multicasting, is discussed in section 6.1 under two fundamental threats, namely the wire tapper adversary and the Byzantine modification. The second parameter, network cost for multicasting with network coding, is considered in section 6.2 with respect to two basic costs, network cost and coding cost.

The proposed solution for a minimum cost secure network coding problem is composed of the following two steps, that is, to:-

1. Identify low cost minimal configurations with lowest wire tapper threats.
2. Examine these minimal configurations for Byzantine modification detection.

The first step is achieved by the evolutionary approach, based on the genetic algorithm. Chapter 4 discusses identification of the minimal configurations $G' \in G$ with optimum network and coding resources, and network code construction based on the minimal configuration identified. The proposed solution in chapter 4 can be directly applied to achieve the first step, if the randomly generated networks in section 4.5 are composed of random sets of the wire tapper adversaries. The solutions against the wire tapper adversaries are prevention techniques, because their detections are extremely complicated. In the solution here, the minimal configurations, G', are categorised as *highly vulnerable* G'_H, or *partially vulnerable* G'_L. The first category can never be protected from wire tappers but simulation results show that a proposed GA based heuristic[7] solution has great potential for identifying the second category. An interesting point of the proposed solution is that the optimisation process is performed without introducing the objective function to the wire tapper adversary because the adversary cannot be physically detected. Moreover, the *partially vulnerable* configurations can be perfectly protected by a proposed coding scheme based on random linear network coding [2] without random packet mixing; this is a significant advantage for minimising the network and coding cost.

This work has addressed single source multicast problems and the proposed GA based centralised algorithm is implemented at the source node. It is guaranteed that the proposed solution previously described allows the source to identify low cost minimal configurations for its multicast transmission. However, the source does not have a way to verify the partial vulnerability of any low cost minimal configuration that it selects. Fortunately, the selected minimal configuration is either *partially vulnerable* or not, and its Byzantine modifications can be detected by a proposed simple technique. The Byzantine modification detection in the network $G(V,E)$ is

[7] Heuristic is a technique designed for solving a problem quicker when classic method are too slow, or for finding an approximate solution when classic method fail to find any exact solution.

impossible, and a costly method, because all nodes (V) and links (E) do not contribute to each multicast transmission. In identification of the minimal configuration $G'((V' < V), (E' < E))$ contributes to overcome these issues because $G'(V', E')$ consists of a well known certain number of nodes and links. Assume the network $G(V, E)$ is error free but it is threatened by the wire tappers and Byzantine modifications. The source is potent enough to select a low-cost, partially-vulnerable configuration, which it then examines for Byzantine modifications prior to initiation multicast transmission. The source sends an identical set of packets through $G'(V', E')$ as its multicast transmission and the sinks obtain these packets by solving the system equations. Each sink independently obtains its solution and reconstructs the set of packets, which may or may not be identical to what was transmitted. Each sink individually sends back its '*acknowledgement'* to the source, if the set of packet is identical, otherwise '*error*'. If the source has received at least one '*error*' message then $G_L^{'}$ consists of Byzantine modification, and the source continues its examination with the rest of $G'(V', E')$. This method is described in more detail later in the chapter.

6.1 NETWORK SECURITY

Leon Panetta, the US secretary of defence, said recently[8] that "the reality is that there is the cyber capability to basically bring down our power grid…to paralyse our financial system in this country to virtually paralyse our country." Moreover Army Gen. Keith B. Alexander, US cyber commander told participants in an American

[8] Speaking at the Business Executives for National Security dinner, New York, October 2012

Enterprise Institute seminar titled “Cybersecurity and American power”[9] that the capabilities exist today for destructive cyber attacks against critical infrastructures. He stated that “An attacker may originate in a country or a criminal gang within the country. It does not matter who did it, you still lose your financial sector or the power grid or systems capabilities for a period of time.” Their statements pledge that the cyber world is essentially protected from cyber attacks for consistency of the future generations.

Network coding is an elegant technique to protect naturally (i.e. without additional security mechanisms) multicast data against the wire tappers. Linearly combined packets, instead of uncoded data, naturally create a multipath diversity against wire tapper adversaries. For example, consider the simple networks in Figure 6-1, consisting of two parallel unit-capacity channels. There are two independent unit-rate information sources (e.g. two movies) located at node *A*, and they intend to securely multicast their data $\{x_1$ and $x_2\}$ to legitimate users (who have paid to receive the information from both sources) at node *D*. Assume the eavesdroppers in the network are able to tap only one link during a time slot and they do not mutually share the received information. Figure 6-1(a) does not show multi path diversity, the independent symbols $\{x_1$ and $x_2\}$ are sent uncoded, and each adversary is able intercept one of them. Figure 6-1(b) shows how linearly combined symbols (over some finite field) are sent via the different routes, but each wire tapper is unable to retrieve any part of the data. The interesting point is that Figure 6-1(a) is defenceless

[9] Washington DC, July 2012

against the wire tappers, but its node *D* prevents decoding complexity, whereas the protected Figure 6-1(b) pays some complexity price.

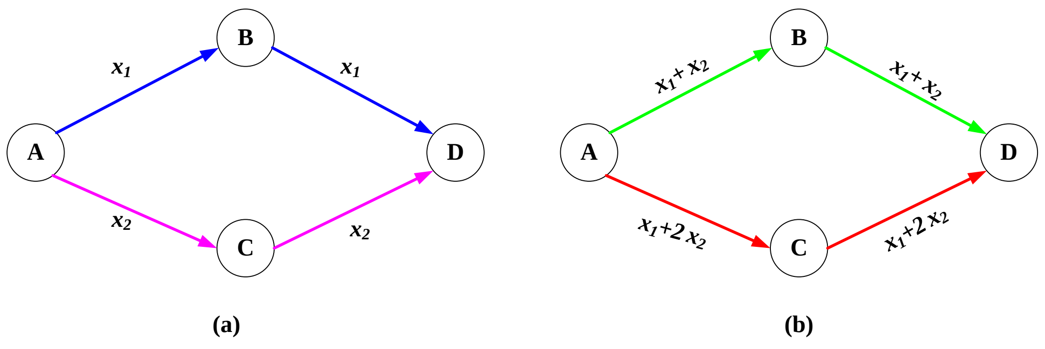


Figure 6-1: (a) uncoded packets are unprotected by the wire tappers; (b) coded packets offer a natural protection against wiretapping.

Considering further that the network in Figure 6-1 consists of not only the eavesdroppers but also a Byzantine attacker at node *B* that performs a jamming attack. Assume node *B* in Figure 6-1(a) modifies symbol $\{x_1\}$ to symbol $\{x_3\}$ and in Figure 6-1(b) modifies $\{x_{1+}x_2\}$ to symbol $\{x_4 + x_5\}$. The legitimate users at node *D* in Figure 6-1(a) are still able to receive symbol $\{x_2\}$, error free, but at node *D* in Figure 6-1(b) decoding errors occur and the users are unable to obtain both symbols. Therefore mixing information streams is harmful if the network consists of the Byzantine attackers.

6.1.1 WIRE TAPPER ADVERSARY

The information theoretically secure concept was introduced by Shannon [3] and concerns two channels which are defined as "*public*" and "*secure*" for the theoretically secure data transmission. Suppose a sender expects to send the output of random source message Ψ with alphabet $A = \{0,1.....(p-1)\}$ to a receiver. The sender can send information via the public channel, whose output can be accessed by the receiver as well as the wiretapper who tries to obtain some information about Ψ, or the sender can send information via the secure channel, whose output can be accessed only by the receiver. The usual way to protect Ψ from the wiretapper is that the sender generates a "secret key" *K* independent of the source message Ψ and uniformly distributed over *A*. Letting *m* be the outcome of Ψ and *k* the outcome of *K*, the sender sends the key *k* to the receiver via the secure channel, and sends $m+k$ (mod *p*) via the public channel. The receiver can obtain *m* by performing a mathematical manipulation on *m* and $m+k$. If the wiretapper taps either the secure channel or the public channel, it is unable to obtain any *m* by knowing either $m+k$ or *k*. The major disadvantage of the Shannon cipher system is that if the wiretapper is able to tap both the channels simultaneously, *m* may be obtained.

However, the Shannon cipher system can be improved by designing the secure codes shown in Figure 6-2 with two linear disjoint paths from the source. Consequently the wiretapper cannot obtain any *m* by knowing either *k* or $k + m$ or $k - m$.

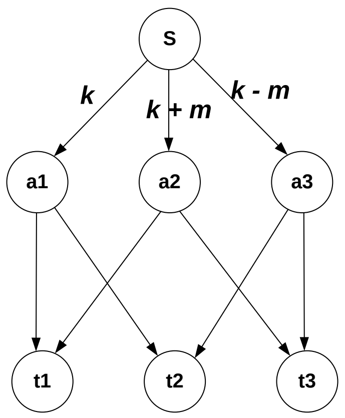


Figure 6-2: A wiretap network representing the (2, 3) – threshold secret sharing scheme.

Based on the observations above and the butterfly network coding model [4], Cai and Yeung [5] proposed a secure network coding model which is called the *wiretap network.*

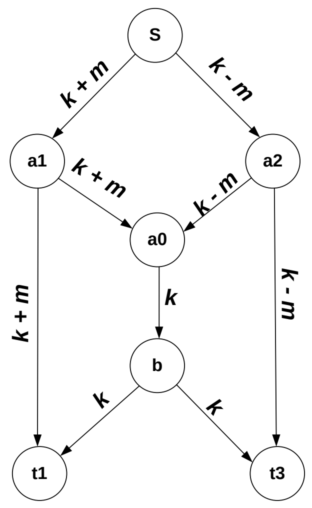


Figure 6-3: Admissible codes for a wiretap network.

This is shown in Figure 6-3 with admissible codes, and consists of a communication network and a collection of subsets of wiretap channels. Any link is susceptible to the wiretapper adversaries and the admissible codes protect the source message *m* from them. Moreover, the admissible codes allow legal users t_1, and t_3 to obtain *m* without any errors. In the [5], a definition of the *wiretap network* (*WN)* was modified such that the collection of wiretap subsets is all subset of channels with cardinalities no larger than *r*, designated *r-WN.* A network code is *r*-secure if it is secure for an *r-WN.* That is, for an *r*-secure network code, a wiretapper can obtain no information about the secure message by accessing any *r-channels.* For example, the Shannon cipher system is the simplest 1-secure network code. Moreover the wiretapper cannot obtain information about the secure message by accessing any 1-*channel.* This basic model provides a key inspiration to build the proposed security scheme here to protect against wiretappers in multicast networks. It is clear that for the existence of r-secure network codes, it is necessary that r is strictly smaller than the value of maximum flow from the source node to every sink node. This is because otherwise a wiretapper accessing all the channels at a minimum cut between the source node and a sink node would have all the information received by the sink node, and therefore could correctly decode the secure message. The proposed security scheme minimizes the size of the *r-channels* using a GA-based heuristic method and the security of the *r-WN* is strengthened by the proposed coding scheme and source packet forwarding technique at an output buffer of the source.

The r-secure linear network code was strengthened to the *strongly r-secure linear network code* by Harada and Yamamoto [6]. For such a network code, a wiretapper can obtain no information about any S components of the source message by accessing $n - s$ channels, provided that the maximum flows to all the sink nodes are at least n, where $s \leq n - r$. They presented a polynomial-time algorithm to construct strongly secure linear network codes. In [7], Cai showed that a random linear network code [8] is strongly secure with high probability, provided that the order of the coding field is sufficiently large.

Jain [9] focused on the relation between security and network topology. In his model there is a single source node and a single sink node in the network, and the entire node may generate random packets to help secure transmission. The model means that messages can be transmitted with perfect security and there is no consideration of the cost incurred. The trade-off between security and the cost of network coding was studied by Tan and Médard [1]. In their model there is a probability that each channel may be accessed by a wiretapper who is interested in the messages from a subset of sources. Their criterion of security is the probability that the wiretapper is able to decode the message of interest correctly. They proposed two heuristic solutions and compared their performance with traditional routing by simulation. Their results showed that network coding may be more effective for both reducing the cost and increasing the security. In the above literature, security is measured by information theoretic quantities (mutual information or entropy) or decoding probability.

6.1.2 BYZANTINE MODIFICATION

Byzantine modification is a fatal threat in network coding multicast because network coding allows the intermediate nodes to mix information. Therefore network coding not only offers benefits but also it unintentionally allows fatal errors. A malicious node usually pretends to forward packets from source to sink, while in reality it injects corrupted packets into the information flow. Since network coding makes the coded packets at the routers, a single corrupted packet can cause a fatal disruption to the decoding operations at the sinks. Therefore this problem is essentially solved unless network coding may perform much worse than traditional network routing in the presence of adversaries.

A few papers have studied the interplay of network coding and Byzantine adversaries, and some of them focus on the detection of the presence of the adversaries[11], others correct the errors which adversaries inject under specific conditions [12],[13]. Ho *et al.*'s approach against Byzantine adversaries [11] is based on distributed randomised network coding [8]. In it, a packet-based random network coding scheme is used, where source nodes include in each source packet some hash symbols calculated as simple polynomial functions of the source data. The sinks check the data and hash values of their decoded packets to determine if modifications have been introduced. This approach requires minimal additional computation because it does not included cryptographic hash functions[10]. The essential condition of the method is that sinks obtain one or more unmodified packets, their contents are unknown when the Byzantine attacker modifies them. These unmodified packets are termed *good*. For example, consider a set of *s* source packets and *g* good packets, which the source packets are multicast using distributed randomized network coding in the finite field $\mathbb{F}_q$. The decoding process performs on *g* good packets and $s - g$ modified packets.

Jaggi *et al.*[14] introduce *the first distributed polynomial-time rate-optimal* network codes that work in the presence of Byzantine nodes. Their algorithms concern adversaries with different attacking capabilities. When the adversary can eavesdrop on all links and jam z_0 links, their first algorithm achieves a rate on C-$2z_0$, where *C* is the maximum multicast rate. In contrast, when the adversary has limited snooping capabilities, their algorithms achieve the higher rate of C-z_0. They are information theoretically secure and operate in a distributed manner, assume no knowledge of the topology, and can be designed and implemented in polynomial time. The decoding process of the system equation is infeasible except by employing redundancy. The adversary injects packets and coding nodes combine them and original source packet with random coefficients. However the system equation

[10] A hash function takes an input and return an output based on that input and it is a one-way function.

becomes extremely complicated as the number of unknowns grows. To address this fatal situation, the source needs to add redundancy to its transmitted packets. Although this is vital to overcome the issue in the decoding process redundancy exhausts network and coding resources (cost).

6.2 COST CRITERION

The cost calculation is considered as a basic function of resource allocation for a unit packet successfully delivered from the source to a set of sinks during a unit time period. The resources required for the process are split into two categories, namely *network* resources and *coding* resources. The first of these resource categories contains costs associated with setup complexity [10], which implies the complexity associated with the computation of the minimal subgraph $G' \in G$ needed to provide the connections. For instance, in routed networks a cheapest cost approach to a unicast connection usually selects a single path. However, when the connection is to be resilient against wiretapping, multiple disjoint paths may be used, which may increase the network cost [15], [16]. While the minimum cost multicast problem in routed networks requires the finding of a directed Steiner tree (NP-hard), the same problem in coded networks can be solved by a linear program in polynomial time [17]; implementation is also possible in a decentralized manner [15]. Moreover, simulation results have shown that network coding can provide multicast connections at a lower cost than traditional routing [15], [18].

The second category (coding) includes buffer memory allocations and the computational power of each coding node and sink node [10]. The computational power of the multicast network is degraded by computational complexity and the coding resources are exhausted, consequently the coding cost is affected by the computational complexity. These issues have been discussed at length in Chapter 5.

The majority of the security schemes to date have been fully focused on improving the quality of the security against wiretappers and Byzantine attackers without considering the network and coding cost. Moreover, the schemes have been highly concerned with the ability to decode the source messages at all sinks with zero errors rather than the impact of randomness on the channel capacity. Utilizing randomness to protect the source messages from wiretappers is effective but contributes to exhaustion of the channel capacity and an increase in codec complexity at the coding and sink nodes. Consequently, the network and coding cost is greatly increased by approaching randomness, but the transmission in the network has to be randomized because otherwise a channel output would be either a function depending on the messages, or simply a constant. Therefore the proposed secure scheme here introduces a method to modify packet forwarding at a source output buffer instead of employing randomness.

6.3 THE PROBLEM AND ITS CONTEXT

Considers a communications network represented by a directed acyclic graph $G=(V,E)$ with unit capacity edges, with the min-cut between the source node and each of the receivers of h. There is a set of h unit rate information sources $\{S_1,S_2.......S_h\}$ and a set of N receivers $\{t_1,t_2.......t_N\}$. The source, S_i, emits σ_i which is an element of some finite field, $\mathbb{F}_q$. In random linear network coding, intermediate nodes linearly combine source symbols with random coefficients, and coded symbols are elements of some finite field, $\mathbb{F}_q$. Moreover each sink is able to communicate reversibly with the source and it sends an acknowledgement of multicast data, which then can be perfectly decoded by Gaussian elimination.

The network is considered to be delay free and error free; the former implies that all nodes simultaneously receive all their inputs and produce their outputs and the later conveys that all packets are routed with zero errors when adversaries are not intercepting. Moreover here it is assumed that each receiver has at least a single set of *h-Linear Disjoint Paths* which are denoted by $(S_i, t_j), 1 \le i \le h, 1 \le j \le N$, from the source to the receiver node j.

The choice of paths is not unique. The object of interest is the minimal subgraphs *G'* of *G*, consisting of the *hN* paths. Moreover the minimal subgraph *G'* is able to show lower network and coding cost during its multicasting, and potentially it should be *partially vulnerable* because it can be perfectly protected against adversaries (wiretapping and Byzantine modification) by the proposed method (coding scheme and technique).

The communication system is introduced on adversaries (CSA); a CSA consists of a network, a collection *W* of wire tapped subsets of channels and a collection *B* of malicious nodes. A set of independent wire tappers (with knowledge of the coding employed) is denoted $\{A_1, A_2 ......A_i\}$, i is any unknown positive integer. A wiretapper, A_i, can arbitrarily choose one but only one wiretap subset $w \in W$, and fully access (the output of) all the channels in w. The communicators over a CSA know the collection *W* of wiretap subsets, but do not know which subset w is chosen by the wiretapper and have no detection method to identify it.

Each *w* consists of an *r*-subset of channels, where *r* can vary arbitrarily. G' consists of sets (S_i ,t_j) of *h-Linear Disjoint Paths* (*h-LDP*), and one or more of the *h-LDP*s may be laid through *w* and A_i may selects that wiretap. The set of wiretappers $\{A_1, A_2......A_i\}$ is interested in a wiretap set $\{w_1, w_2......w_i\}$ in G' and the wiretap set $\{w_1, w_2......w_i\}$ consists of an *r*-subset $\{r_1, r_2......r_i\}$ of the channels consecutively. If the r_i – subset is located in the sets (S_i, t_j) of *h-LPD*s and the condition $|r_i| < h$ is satisfied then the subgraph G' is partially vulnerable, otherwise the condition $|r_i| \geq h$ is satisfied and the subgraph G' is highly vulnerable. When A_i satisfies the latter condition (highly vulnerable subgraph $G_H^{'}$) it can obtain the source messages $\{S_1, S_2......S_h\}$ with zero errors by solving the *h*-linear equations by Gaussian elimination, implying that A_i is authorised as a legal sink node t_j in the network for $\{S_1, S_2......S_h\}$. When G' is partially vulnerable, this implies that the wiretapper A_i is able to tap any single channel (or channels less than *h*) of the r_i-subset, and, if that channel is transmitting uncoded packets of the data stream S_i, then the wiretapper A_i can obtain S_i with zero errors. Therefore the random linear-coding based scheme provides a solution that strengthens the protection of the partially vulnerable subgraph $G_L^{'}$. Moreover, the proposed packet forwarding technique provides significant protection to $G_L^{'}$ except using the costly random packets.

Moreover a set of Byzantine attackers (independently performed) is denoted by $\{B_1, B_2......B_j\}$, where *j* is unknown positive integer. A Byzantine attacker, B_j, can arbitrarily select a node (call a malicious node) but only one node $v \in V$ except the source or sinks, and all outgoing packets of the malicious node are modified without changing their packet size.

Chapter 4 defined the cost calculation process more firmly, and the simulation results therein showed that the proposed algorithmic solution has a strong potential to identify the low cost minimal subgraph *G′*. Moreover, it also explained network code construction based on the sparse matrix of *G′*. The simulation in this chapter is a modification of that of chapter 4, because the randomly-generated networks for each project are additionally introduced by adversaries. The GA searches to identify $G_L^{'}$s, and, once identified, these are examined to detect the malicious nodes.

To illustrate the issues, the network shown in Figure 6-4 is considered. The source *S* wishes to multicast three data streams $\{a^{s_1}, b^{s_2}, c^{s_3}\}$ to a set of sinks t_1, t_2 and t_3. The example link costs ($\xi_{(v,u)}$) are indicated on the figure. There are three wiretap sub sets $\{w_1, w_2, w_3\}$ and one malicious node {7} shown on the network. Figure 6-4 shows, two wiretappers $\{A_1, A_2\}$ who select the wiretap subsets $\{w_2, w_3\}$, and one Byzantine attacker (B_1) selects the malicious node {7}. First, prior initiating multicast transmission, the source expects to identify the minimum cost, partially protected multicast subgraph $G_L^{'}$ in the network ***G*.**

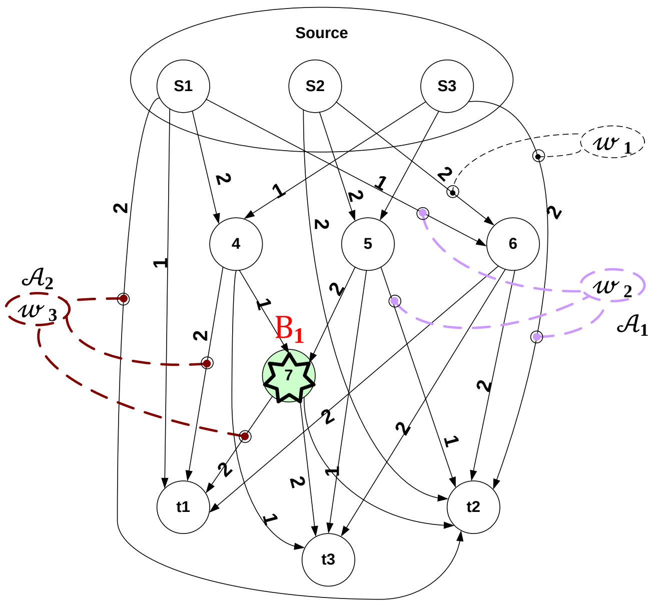


Figure 6-4: The communication system on adversaries to illustrate the problem and proposed method.

Figure 6-5 (a) and (b) show two different low cost minimal subgraphs ***G′*** in the network ***G*** of Figure 6-4, assuming that the ***G′***'s were identified by the proposed GA based method. Each sink t_1, t_2, t_3 belongs to the two different subsets of the 3- *linear disjoint paths* in each of Figure 6-5 (a) and Figure 6-5 (b). In Figure 6-5 (a), the wiretap subset w_1 is laid on the 1- *linear disjoint path* $\langle S_3, t_2 \rangle$, w_2 is laid on 1-*linear disjoint path* $\langle S_1, 6, t_3 \rangle$ and the set of 2-*linear disjoint paths* $(\langle S_2, 5, t_2 \rangle, \langle S_3, t_2 \rangle)$ and w_3 is laid on the set of 1-*linear disjoint path* $\langle S_1, t_2 \rangle$ and the set of 2-*linear disjoint paths* $(\langle S_2, 5, 7, t_1 \rangle, \langle S_3, 4, t_1 \rangle)$. However, during the multicast transmission session, the wiretap subset w_1 is not selected by the wiretappers A_1 or A_2 and the wiretap subset w_2 is selected by the wiretapper A_1 at links $(S_1, 6), (5, t_2), (S_3, t_2)$. These links do not originate at coding nodes and A_1 can easily obtain $\{a^{S_1}, b^{S_2}, c^{S_3}\}$. A_2 selects wiretap subset w_3 at links $(S_1, t_2), (4, t_1), (7, t_1)$, but link $(7, t_1)$ originates at coding node 7 and

A_2 has to make an additional effort rather than A_1 to solve the system equations via Gaussian elimination. Therefore the subgraph G' in Figure 6-5 (a) is defined as highly vulnerable ($G_H^{'}$) and cannot be protected using the multicast network coding technique.

In Figure 6-5 (b), the wiretap subset w_2 is laid on the 2-*linear disjoint paths* $(\langle S_1, 6, t_2 \rangle, \langle S_3, t_2 \rangle)$. The wiretapper A_1 is unable to form three linear equations tapping the 2-*linear disjoint paths*, and source data $\{a^{s_1}, b^{s_2}, c^{s_3}\}$ cannot be obtained by Gaussian elimination. It is possible that A_1 can obtain the source data $\{a^{s_1}, c^{s_3}\}$ because the tapped links $(S_1, 6), (S_3, t_2)$ are not naturally protected by the multicast network coding technique. Moreover, the wiretap subset w_3 is laid on the 1-*linear disjoint path* $\langle S_3, 4, t_1 \rangle$, and A_2 cannot obtain the source data by Gaussian elimination. Also the tapped link $(4, t_1)$ is a coded link at the coding node 4 and the link $(4, t_1)$ is protected. Therefore the subgraph G' is defined as a partially-vulnerable minimal subgraph ($G_L^{'}$) and can be protected by the proposed method (random coding scheme and packet forwarding technique at the source).

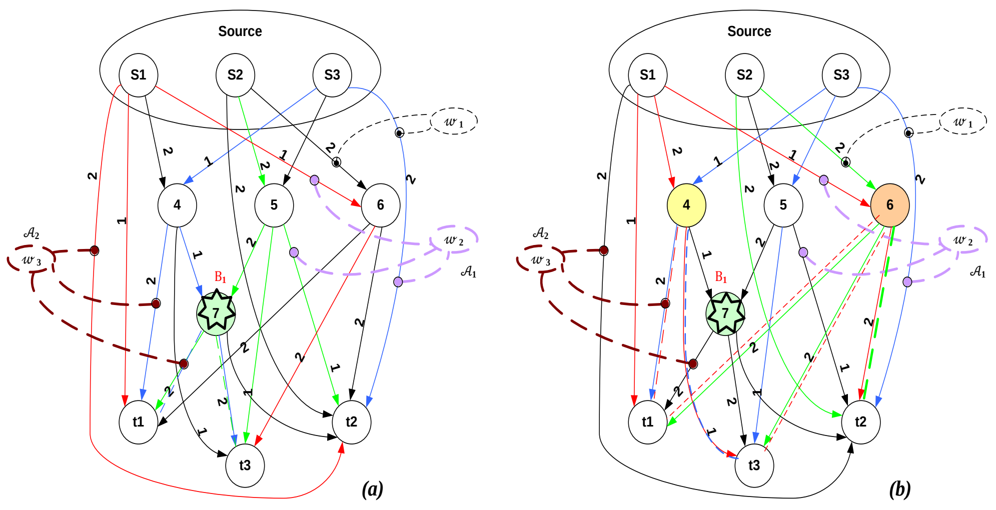


Figure 6-5: (a) Highly vulnerable minimal subgraph *G'*$_H$; (b) partially vulnerable minimal subgraph *G'*$_L$.

Second, prior to initiating multicast transmission, the source examines G' (most probably $G_L^{'}$) for the malicious nodes. Assume the source selects $G_H^{'}$ in Figure 6-5 (a) for its examination and it forwards identical test packets $\{\sigma^{s_1},\sigma^{s_2},\sigma^{s_3}\}$ as its multicast transmission. The Byzantine attacker (B_1) at node 7 randomly modifies its outgoing packets as $\{\alpha^{(s_1,s_3)},\beta^{(s_1,s_3)}\}$. Sinks t_1,t_3 are unable to obtain $\{\sigma^{s_1},\sigma^{s_2},\sigma^{s_3}\}$ and each of them sends a message 'error' to the source. Moreover sink t_2 send its acknowledgement because it obtained $\{\sigma^{s_1},\sigma^{s_2},\sigma^{s_3}\}$. The proposed solution is defined as a theoretically secured network coding scheme, therefore the network is considered error free and the source justifies, $G_H^{'}$ in Figure 6-5 (a) is comprised by the malicious nodes and discarded. Sinks t_1,t_2,t_3 in Figure 6-5 (b) obtain $\{\sigma^{s_1},\sigma^{s_2},\sigma^{s_3}\}$ and each of them sends its acknowledgment; consequently the source selects $G_L^{'}$ for its multicast transmission.

6.4 CODING SCHEME AND PACKET FORWARDING TECHNIQUE AT SOURCE

The GA based method can potentially identify $G_L^{'}$ s, and their malicious nodes can be simply detected by the multicasting of identical test packets (e.g. $\{\sigma^{s_1}, \sigma^{s_2}, \sigma^{s_3}\}$). However the G'_L s are not fully protected from the wiretappers. Links which carry uncoded packets are potentially vulnerable without randomness. Utilizing randomness to protect the source messages from wiretappers is effective but contributes to exhaustion of the channel capacity and an increase in codec complexity at the coding and sink nodes. Consequently, the network cost is greatly increased by approaching randomness but the transmission in the network has to be randomized because otherwise a channel output would be either a function depending on the messages or simply a constant. Here there follows a discussion of how the random coding scheme and packet forwarding technique at the source provides full protection to $G_L^{'}$ except costly randomness.

6.4.1 CODING SCHEME

The proposed coding scheme is based on the linear random coding method [2] and the proposed coding scheme in section 4.2. The source intends to forward its multicast packets through *G'* and coding nodes combine them linearly with random coefficients $(\alpha_i) \in \mathbb{F}_q$. The linear random coding method can minimize the coding complexity of the network coding scenario and thus the network coding cost. This technique naturally introduces path diversity and prevents a channel output as either a function or simply a constant, depending on the messages.

However, a major disadvantage is that the method may generate linearly dependent *local coding vectors* at the coding nodes, meaning that the sinks may be unable to form full rank matrixes as their *global coding vectors.* If at least a sink obtains a non full rank matrix, it causes decoding errors and it must send an error message to the source, which has to transmit the same set of simultaneous packets to all sinks, increasing the complexity and cost. If $\alpha_i s$ are chosen in sufficiently large $\mathbb{F}_q$, then this issue is sufficiently small (example in section 2.1.1).

6.4.2 PACKET FORWARDING TECHNIQUE AT THE SOURCE

This is a prominent part of the low cost security scheme and the technique is introduced here instead of costly randomness. The source in $G_L^{'}$ connected with strictly *h* nodes as its first set of edges. Assume the multicast session is split by τ simultaneous sets of packets $\Phi_{s_i}^{j}(1 \le i \le h; 0 \le j \le \tau - 1)$. During the multicast session each simultaneous set of the packets $\Phi_{s_i}^{j}$ is circularly shifted by a *j*^th^ value. Figure 6-6 shows the packet forwarding technique at the source node. This technique protects all uncoded links in $G_L^{'}$ but does not mix costly random packets. The simulation results show that the wiretappers are confused and unable to obtain any clear message during the multicast session.

$$\begin{array}{ccccc}
\Phi_{s_{h-2}}^{j=\tau-1} & \Phi_{s_{h-1}}^{j=\tau-1} & \cdots & \Phi_{s_1}^{j=\tau-1} & \Phi_{s_2}^{j=\tau-1} \\
\Phi_{s_{h-3}}^{j=\tau-2} & \Phi_{s_{h-2}}^{j=\tau-2} & \cdots & \Phi_{s_h}^{j=\tau-2} & \Phi_{s_1}^{j=\tau-2} \\
\vdots & \vdots & & \vdots & \vdots \\
\Phi_{s_h}^{j=1} & \Phi_{s_1}^{j=1} & \cdots & \Phi_{s_{h-2}}^{j=1} & \Phi_{s_{h-1}}^{j=1} \\
\Phi_{s_1}^{j=0} & \Phi_{s_2}^{j=0} & \cdots & \Phi_{s_{h-1}}^{j=0} & \Phi_{s_h}^{j=0} \\
\Downarrow & \Downarrow & & \Downarrow & \Downarrow
\end{array}$$

Figure 6-6: The packets forwarding technique at the source

6.4.3 SIMULATION PHASE

The implementation of both techniques in section 6.4.1 and 6.4.2 is now discussed, consisting of a virtual network with only basic functions (forwarding and coding) at intermediate nodes. The source node forwards packets with via the technique of section 6.4.2, intermediate nodes follow section 6.4.1, and sinks perform Gaussian elimination to solve the system equations. The wiretap subsets were implemented in the same way as the sinks. Only links and nodes of $G_L^{'}$ were kept 'active' and the remainder were kept 'inactive'. The link 'active' implies that an output buffer of *tail* node can be accessed by an input buffer of *head* node. If any node has two or more active in – links then it can be sensed, it has to perform as coding node and rest of intermediate nodes have to perform as only forwarding nodes. A simple packet format was used here with payload and header; the latter consisted of a source vector or coding vector, packet order number and destination address. The payloads utilised were symbols taken from some $\mathbb{F}_q$. If any coding node detected packets with identical order numbers, then they were combined linearly with random coefficients in some $\mathbb{F}_q$. Each sink obtained the decoding matrix to solve the system equations based on the packet's order number.

6.5 SIMULATION RESULTS AND DISCUSSION

Simulations of the proposed solution consisted of two parts. The first focused on the identification of $G_L^{'}$s. Each GA technique is evaluated based on potential of their identifications, here it is defined as the probability of identifying at least one $G_L^{'}$ (without malicious nodes) during each run $(\Pr_{G_L^{'}}^{(MOGA/VEGA)})$. For example, in Table 6-1, MOGA and VEGA are evaluated as $\Pr_{G_L^{'}}^{(MOGA)}$ and $\Pr_{G_L^{'}}^{(VEGA)}$,and their evaluations are 50% and 80% respectively. These simulations were similar to those in section 4.5 but with the difference here that networks were randomly generated with a level of adversaries.

<table>
<tr><th rowspan="3">Run</th><th colspan="10">Topological detail and level of adversaries
Nodes - 27; Links – 57; Sinks – 07; Wiretap subsets – 03; Malicious nodes - 02</th></tr>
<tr><th colspan="5">MOGA</th><th colspan="5">VEGA</th></tr>
<tr><th>$\lvert G'\rvert$</th><th colspan="2">$\lvert G'_L\rvert$</th><th colspan="2">$\lvert G'_H\rvert$</th><th>$\lvert G'\rvert$</th><th colspan="2">$\lvert G'_L\rvert$</th><th colspan="2">$\lvert G'_H\rvert$</th></tr>
<tr><th></th><th></th><th>With malicious nodes</th><th><b>Without malicious nodes</b></th><th>With malicious nodes</th><th>Without malicious nodes</th><th></th><th>With malicious nodes</th><th><b>Without malicious nodes</b></th><th>With malicious nodes</th><th>Without malicious nodes</th></tr>
<tr><td>1</td><td>4</td><td>1</td><td><b>2</b></td><td>1</td><td>0</td><td>5</td><td>1</td><td><b>2</b></td><td>1</td><td>1</td></tr>
<tr><td>2</td><td>-</td><td>-</td><td><b>-</b></td><td>-</td><td>-</td><td>6</td><td>2</td><td><b>1</b></td><td>3</td><td>0</td></tr>
<tr><td>3</td><td>5</td><td>2</td><td><b>1</b></td><td>2</td><td>0</td><td>4</td><td>1</td><td><b>1</b></td><td>1</td><td>1</td></tr>
<tr><td>4</td><td>4</td><td>1</td><td><b>0</b></td><td>1</td><td>2</td><td>4</td><td>2</td><td><b>1</b></td><td>0</td><td>1</td></tr>
<tr><td>5</td><td>-</td><td>-</td><td><b>-</b></td><td>-</td><td>-</td><td>-</td><td>-</td><td><b>-</b></td><td>-</td><td>-</td></tr>
<tr><td>6</td><td>-</td><td>-</td><td><b>-</b></td><td>-</td><td>-</td><td>5</td><td>1</td><td><b>1</b></td><td>2</td><td>1</td></tr>
<tr><td>7</td><td>4</td><td>1</td><td><b>2</b></td><td>0</td><td>1</td><td>4</td><td>1</td><td><b>2</b></td><td>1</td><td>0</td></tr>
<tr><td>8</td><td>4</td><td>0</td><td><b>1</b></td><td>2</td><td>1</td><td>4</td><td>2</td><td><b>0</b></td><td>1</td><td>1</td></tr>
<tr><td>9</td><td>5</td><td>1</td><td><b>1</b></td><td>1</td><td>2</td><td>5</td><td>1</td><td><b>1</b></td><td>1</td><td>2</td></tr>
<tr><td>10</td><td>-</td><td>-</td><td><b>-</b></td><td>-</td><td>-</td><td>4</td><td>1</td><td><b>1</b></td><td>2</td><td>0</td></tr>
</table>

Table 6-1: Simulation results for project – 1, each run uses a randomly generated acyclic network with a level of adversaries.

In Table 6-1, the hyphen (-) indicates ‘failed search’ and the rest of runs were able to identify at least four low cost minimal subgraphs (G'). These simulations were not performed on actual multicast traffic delivery. Moreover, an objective function was not assigned for the detection of the wiretappers. However, a graphical representation (e.g. Figure 4-21(b)) of the subgraph (G') is used to identify G'_L without the malicious nodes.

In Table 6-2, MOGA represents poor performance for small scale networks but VEGA performs differently. Moreover MOGA shows better performance than VEGA for large scale networks.

Project	Topological detail			Level of adversaries		Probability of identifying at least one $G_L^{'}$ (without malicious nodes) during each run	
	Nodes	Links	Sinks	Wiretap sub sets	Malicious nodes	$\Pr_{G_L^{'}}^{(MOGA)}$	$\Pr_{G_L^{'}}^{(VEGA)}$
1	27	57	07	03	02	50%	80%
2	30	68	07	04	03	60%	90%
3	35	92	12	05	04	66.67%	60%
4	40	113	17	06	05	70%	53.33%

Table 6-2: Simulation results analysis for all projects

The second part of simulations investigates how $G_L^{'}$ (without malicious nodes) is perfectly protected by the proposed methods in section 6.4.1 and 6.4.2. This simulation used $G_L^{'}$ in Figure 6-5 (b). The payload of each packet is a symbol of the finite field $\mathbb{F}_{2^{32}}$, and the random coefficients are selected by $\mathbb{F}_{2^4}$. The first row of Figure 6-7 shows, multicast data transmitted from $\{S_1, S_2, S_3\}$. The second, third and forth rows of Figure 6-7show that sinks $\{t_1, t_2, t_3\}$ obtained source data via the Gaussian elimination. The row 5 of Figure 6-7 shows, wiretappers (A_2,A_1) obtained data. A_2 tapped only coded link $(4, t_1)$ and thus could not form the system equations. A_1 tapped two uncoded links $(S_1, 6), (S_3, t_2)$ but it could not obtain the source data because the proposed techniques in section 6.4.1 and 6.4.2 interrupted the tapping without additional cost or effort.

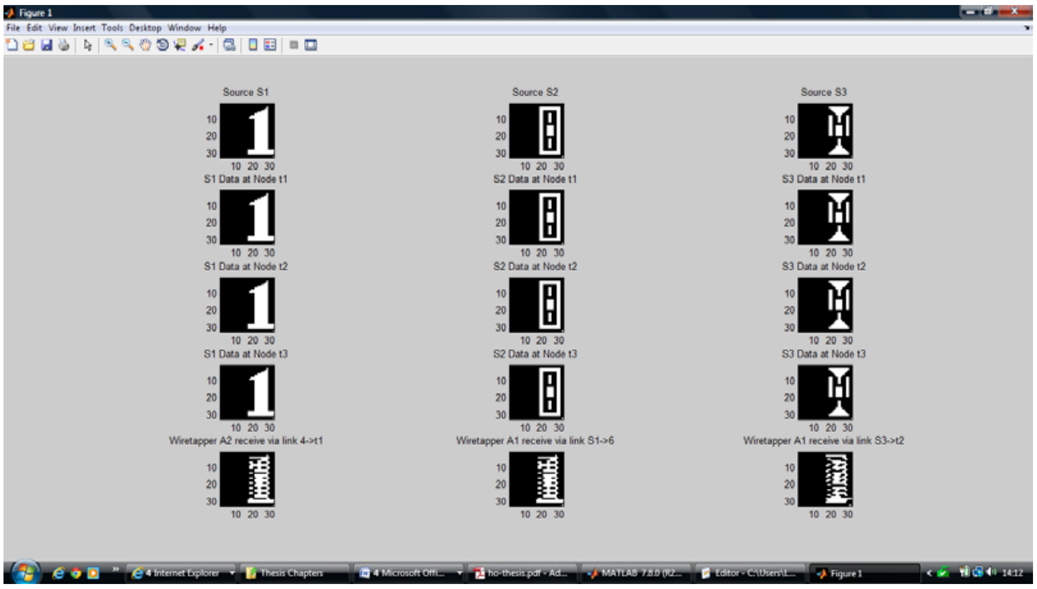


Figure 6-7: Simulation results analysis, how G'_L is fully protected by the proposed techniques.

6.6 CONCLUSION

This has been a first attempt to investigate jointly network cost, coding cost, wiretapper adversary, and Byzantine modification in the multicast scenario. The proposed solution expects to identify low cost (network cost and coding cost) minimal configurations (G') in the adversary network, which is categorised as NP-hard; an evolutionary approach is essential to solve it. These *G'* are classified as highly vulnerable minimal configurations (G'_H) and lower vulnerable minimal configurations (G'_L). The G'_Ls can only be protected from the wiretappers because each wiretap subset does not cover h linear disjoint paths. However G'_Ls cannot be identified straightforwardly because the wiretapper adversaries cannot be detected. Therefore, the proposed solution is heuristic and the simulation results show that it has good potential to identify G'_Ls. The network is assumed to be error free, except the errors are caused by the adversaries, and G'_Ls are examined for malicious nodes.

A certain number of nodes (only nodes in $G_L^{'}$) is considered for this examination instead of all nodes in *G*.

The random coding scheme and packet forwarding technique at the source have good potential to provide optimum uncertainty to data on paths and paths diversity. These techniques affect the wiretappers in $G_L^{'}$.

The first part of simulation results shows that VEGA performance is better than MOGA for small scale networks, and vice versa for large scale networks. The second part shows that any $G_L^{'}$ can be perfectly protected by the proposed random coding and packet forwarding technique at the source without costly randomness.

6.7 REFERENCES

[1] J. Tan and M. Médard, “Secure network coding with a cost criterion”, Proc. of the 4th Inter Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2006.

[2] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, “A random linear network coding approach to multicast”, IEEE Trans. on Inform. Theory, vol. 52, pp. 4413-4430, Oct. 2006.

[3] C. E. Shannon, “Communication theory of secrecy systems”, Bell Sys. Tech. Journal 28, pp. 656-715, 1949.

[4] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, “Network information flow,” IEEE Trans. Info. Theory, IT- 46: 1204-1216, 2000.

[5] N. Cai and R. W. Yeung, “Secure network coding,” IEEE International Symposium on Information Theory, Lausanne, Switzerland, Jun 30-Jul 5, 2002.

[6] K. Harada and H. Yamamoto, “Strongly Secure Linear Network Coding,” EICE Transactions on Fundamentals, vol. E91-A, no.10, pp. 2720-2728, 2008.

[7] N. Cai, “Valuable Messages and Random Outputs of Channels in Linear Network Coding,” IEEE International Symposium on Information Theory 2009, Seoul, Korea, June 28-July 3, 2009.

[8] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, A random linear network coding approach to multicast, IEEE Trans. on Inform. Theory, vol. 52, pp. 4413-4430, Oct. 2006.

[9] K. Jain, Security based on network topology against the wiretapping attack, IEEE Wireless Communications, vol. 11, no. 1, pp. 68-71, 2004.

[10] C. Fragouli and E. Soljanin, “Network Coding Fundamentals”, Foundations and Trends in Networking, vol. 2, no. 1, pp. 1-133, 2007.

[11] T. C. Ho, B. Leong, R. Koetter, M. Médard, M. Effros, and D. R. Karger. Byzantine modification detection in multicast networks using randomized network coding. In *International Symposium on Information Theory*, 2004.

[12] S. Jaggi, M. Langberg, T. Ho, and M. Effros. Correction of adversarial errors in networks. In *Proceedings of International Symposium in Information Theory and its Applications*, Adelaide, Australia, 2005.

[13] C. Gkantsidis and P. Rodriguez. Cooperative Security for Network Coding File Distribution. In *IEEE INFOCOM*, 2006.

[14] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Médard. Resilient network coding in the presence of Byzantine adversaries. *IEEE Transactions on Information Theory,* 54(6): 2596 – 2603, June 2008.

[15] D. S. Lun, N. Ratnakar, R. Koetter, M. Médard, E. Ahmed, and H. Lee, “Achieving minimum-cost multicast: A decentralized approach based on network coding,” Proc. - IEEE INFOCOM, Mar. 2005.

[16] J. Jeong, and J. Ma, “Security analysis of multi-path routing scheme in ad hoc networks,” Lecture Notes in Computer Science, vol. 3320, pp.694 - 697, 2004.

[17] D. S. Lun, M. M´edard, T. Ho, and R. Koetter, “Network coding with a cost criterion,” IEEE International Symposium on Information Theory and its Applications, 2004.

[18] P. A. Chou, Y. Wu, and K. Jain, “Practical network coding,” Proc. 41st Annual Allerton Conference on Communication, Control, and Computing, Oct. 2003.

APPENDIX A …

1 FULL RANK MATRIX

Definition: When all of the vectors in a matrix are linearly independent, the matrix is said to be **full rank**. Consider the matrices **A** and **B** below.

$$A=\begin{bmatrix}1 & 2 & 3\\ 2 & 4 & 6\end{bmatrix} \quad B=\begin{bmatrix}1 & 0 & 2\\ 2 & 1 & 0\\ 3 & 2 & 1\end{bmatrix}$$

Notice that row 2 of matrix **A** is a scalar multiple of row 1; that is, row 2 is equal to twice row 1. Therefore, rows 1 and 2 are linearly dependent. Matrix **A** has only one linearly independent row, so its rank is 1. Hence, matrix **A** is not full rank.

Now, look at matrix **B**. All of its rows are linearly independent, so the rank of matrix **B** is 3. Matrix **B** is full rank.

2 GAUSSIAN ELIMINATION

Gaussian elimination is a method for solving matrix equations of the form

$$AX = b$$

To perform Gaussian elimination starting with the system of equations

$$\begin{bmatrix}a_{11} & a_{12} & \cdots & a_{1k}\\ a_{21} & a_{22} & \cdots & a_{2k}\\ \vdots & \vdots & \ddots & \vdots\\ a_{k1} & a_{k2} & \cdots & a_{kk}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ \vdots\\ x_k\end{bmatrix}=\begin{bmatrix}b_1\\ b_2\\ \vdots\\ b_k\end{bmatrix}$$

Compose the “augmented matrix equation”

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} & \vert b_1 \\ a_{21} & a_{22} & \cdots & a_{2k} & \vert b_2 \\ \vdots & \vdots & \ddots & \vdots & \\ a_{k1} & a_{k2} & \cdots & a_{kk} & \vert b_k \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}$$

Here, the <u>column vectors</u> in the variables **X** is carried along for labelling the matrix rows. Now, perform <u>elementary row operations</u> to put the augmented matrix into the <u>upper triangular</u> form

$$\begin{bmatrix} a'_{11} & a'_{12} & \cdots & a'_{1k} & \vert b'_1 \\ 0 & a'_{22} & \cdots & a'_{2k} & \vert b'_2 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \cdots & a'_{kk} & \vert b'_k \end{bmatrix}$$

Solve the equation of the k^{th} row for x_k then substitute back into the equation of the $(k-1)^{st}$ row to obtain a solution for x_{k-1}, etc., according to the formula

$$x_i = \frac{1}{a'_{ii}} \left(b'_i - \sum_{j=i+1}^{k} a'_{ij} x_j \right)$$

3 BINARY FIELD

Example (*binary field* $\mathbb{F}_{2^4}$) The elements of $\mathbb{F}_{2^4}$ are the 16 binary polynomials of degree at most 3:

0	Z^2	Z^3	Z^3+Z^2
1	Z^2+1	Z^3+1	Z^3+Z^2+1
Z	Z^2+Z	Z^3+Z	Z^3+Z^2+Z
$Z+1$	Z^2+Z+1	Z^3+Z+1	Z^3+Z^2+Z+1

The followings are some examples of arithmetic operations in $\mathbb{F}_{2^4}$ with reduction polynomial $f(z)=Z^4+Z+1$.

1. Addition: $(Z^3+Z^2+1)+(Z^2+Z+1)=Z^3+(1+1)Z^2+Z+(1+1)=Z^3+Z$
2. Subtraction:
   $(Z^3+Z^2+1)-(Z^2+Z+1)=Z^3+(1+(-1))Z^2+Z+(1+(-1))=Z^3+Z$
   (Note that since -1=1 in $\mathbb{F}_2$, we have $-a=a$ for all $a\in\mathbb{F}_{2^k}$).
3. Multiplication: $(Z^3+Z^2+1).(Z^2+Z+1)=Z^2+1$ since
   $(Z^3+Z^2+1).(Z^2+Z+1)=Z^5+Z+1$ and
   (Z^5+Z+1) mod $(Z^4+Z+1)=Z^2+1$
   $$\begin{array}{r|l} & \overline{Z\phantom{{}+Z^2+Z+1}} \\ (Z^4+Z+1) & Z^5 + + Z+1 \\ & \underline{Z^5+Z^2+Z\phantom{{}+1}} \\ & \phantom{Z^5+{}}Z^2+1 \end{array}$$
4. Inversion: $(Z^3+Z^2+1)^{-1}=Z^2$ since
   $(Z^3+Z^2+1)^{-1}.Z^2$ mod $Z^4+Z+1=1$

3.1 MODULO – 2 OPERATIONS

$\oplus$	0	1
0	0	1
1	1	0

Modulo – 2 Addition (XOR)

.	0	1
0	0	0
1	0	1

Modulo – 2 Multiplication (AND)

4 ALGORITHM ANALYSIS

4.1 POLYNOMIAL TIME

Definition: An algorithm is said to be solvable in polynomial time if the number of steps required to complete the algorithm for a given input size n is $O(n^k)$ for some nonnegative integer k.

i.e. An algorithm $\mathfrak{A}$ is polynomial time if $time_{\mathfrak{A}}(n) \leq f(n)$ and f is a polynomial $(f = a_1 n^k + a_2 n^{k-1} + ... + a_m)$.

Polynomial time algorithms are said to be "fast". Mathematical operations such as addition, subtraction, multiplication, division can be performed in polynomial time. Computing the digits of mathematical constant, including π can also be done in polynomial time.

4.2 GREEDY ALGORITHM

Definition: An algorithm that always takes the best immediate, or local, solution while finding an answer. Greedy algorithms can find the overall, or globally, *optimal solution* for some *optimisation problems.*

Greedy algorithms are usually quicker because they do not consider the details of possible alternatives. Example Prim-Jarnik algorithm is used to compute a *minimum spanning tree.* The Linear Information Flow algorithm is used to network code design.

APPENDIX B ...

1 MATLAB CODES TO CREATE A FUNCTION FILE TO SEQUENTIALLY RUN THE ALGORITHMS

```
%% A MATLAB programme for run 'myDir' files

%  Lalith P. Karunarathne, BEng(Hons)
%  University Of Warwick, Coventry
%*****************************************************
function RunFirst_Res()

myDir = 'C:\Users\LALITHK\Documents\MATLAB\First_ResearchPaper';

d = dir([myDir filesep '*.m']);
for jj=1:numel(d)
    try
        toRun = fullfile(myDir, d(jj).name);
        fprintf('Running "%s"', toRun);
        run(toRun)
    catch E
        % Up to you!
    end
end
```

2 MATLAB CODES FOR CREATING AND VIEWING A RANDOM ACYCLIC NETWORK

```
%%
********************************************************************
%  Random Acyclic Network Creation

%  Lalith P. Karunarathne, BEng(Hons)
%  University Of Warwick, Coventry
%%
********************************************************************
****
%  Acyclic Network Creation with Sparse Matrix

NumNodes = 27; % Define A number of Nodes (V) in the Random Network

Nd_Lk_Mtx = zeros(NumNodes,NumNodes);% Create Zero matrix with size
%|V|*|V|
Nd_Lk_Mtx(1,[4 6 14 22])=1;
Nd_Lk_Mtx(2,[5 8 10 14 23])=1;
Nd_Lk_Mtx(3,[7 9 16 21])=1;
Nd_Lk_Mtx(4,[10 11])=1;
Nd_Lk_Mtx(5,[11 12])=1;
Nd_Lk_Mtx(6,[13 25])=1;
Nd_Lk_Mtx(7,[12 14])=1;
Nd_Lk_Mtx(8,[13 16])=1;
Nd_Lk_Mtx(9,[12 13])=1;
Nd_Lk_Mtx(10,[15 20 24])=1;
Nd_Lk_Mtx(11,[17 19 24])=1;
Nd_Lk_Mtx(12,[15 22 24 25])=1;
Nd_Lk_Mtx(13,[19 25])=1;
Nd_Lk_Mtx(14,[17 18 24])=1;
Nd_Lk_Mtx(15,[21 23])=1;
Nd_Lk_Mtx(16,[18 20 21 22])=1;
Nd_Lk_Mtx(17,[23])=1;
Nd_Lk_Mtx(18,[21 24])=1;
Nd_Lk_Mtx(19,[22])=1;
Nd_Lk_Mtx(20,[23])=1;
Nd_Lk_Mtx([13 16 15],[26])=1;
Nd_Lk_Mtx([1 2 16],[27])=1;

%% MATLAB Codes from Bioinformatics Tool Box for Viewing the Acyclic
Network

bg = biograph(Nd_Lk_Mtx);
h=view(bg);


%% ADD Weight to graph
Weight=[2 1 2 3 1 2 1 3 2 2 3 1 4 1 4 2 3 3 1 4 2 1 3 2 3 2 1 2 1 3
2 1 1 2 3 2 4 1 2 3 2 3 2 3 1 2 2 2 3 3 2 1 2 3 1 2 3 0];
Direct_Graph=sparse([1 1 1 1 2 2 2 2 2 3 3 3 3 4 4 5 5 6 6 7 7 8 8 9
9 10 10 10 11 11 11 12 12 12 12 13 13 14 14 14 15 15 16 16 16 16 17
18 18 19 20 13 16 15 1 2 16 27],...
                    [4 6 14 22 5 8 10 14 23 7 9 16 21 10 11 11 12 13
25 12 14 13 16 12 13 15 20 24 17 19 24 15 22 24 25 19 25 17 18 24 21
23 18 20 21 22 23 21 24 22 23 26 26 26 27 27 27 27],Weight);

%% MATLAB Codes from Bioinformatics Tool Box for Viewing the Direct
with
% weights of its edges

%bg = biograph(Nd_Lk_Mtx);
%h=view(bg);
h = view(biograph(Direct_Graph,[],'ShowWeights','on'));
```

3 MATLAB CODES FOR IDENTIFYING THE SOURCE, SINKS AND CODING NODES IN THE ACYCLIC NETWORK

```
%%
*******************************************************************
 %  The Nodes Classification in the Acyclic Network
 %  Lalith P. Karunarathne, BEng(Hons)
 %  University Of Warwick, Coventry
 %
*******************************************************************
****

tic  %Measure performance using stopwatch timer
```

```
%% Codes for Sinks identification  (Following Graph Theory)
All_Sinks = [];
for i=1:NumNodes
    if all(Nd_Lk_Mtx(i,:)==0)
        All_Sinks(i,:)=i;
    end
end
All_NeWok_Sinks = find(All_Sinks);  %find::function(MATLAB)
Test_All_Net_Sink = zeros(1,NumNodes);
for i=1:length(All_NeWok_Sinks)
    Test_All_Net_Sink(1,All_NeWok_Sinks(i))=1;
end

%% Codes for Source Nodes identification (Following Graph Theory)
All_Sources = [];
for j=1:NumNodes
    if all(Nd_Lk_Mtx(:,j)==0)
        All_Sources(:,j)=j;
    end
end
All_NeWok_Sources = find(All_Sources);
Test_All_Netw_Sour=zeros(1,NumNodes);
for i=1:length(All_NeWok_Sources)
    Test_All_Netw_Sour(1,All_NeWok_Sources(i))=1;
end

%% Codes for Coding Nodes Identification
 %Identify coding & Sinks Nodes together
Mer_Sink_Nod=zeros(1,NumNodes);
for i=1:NumNodes
    [Row2,Col2]=find(Nd_Lk_Mtx(:,i));
    if length(Row2)>=2
        Mer_Sink_Nod(1,i)=1;
    end
end
[Row3,Col3]=ind2sub(size(Mer_Sink_Nod),find(Mer_Sink_Nod(1,:)==1));

%% Separation the Merging nodes from Sinks
Merg_Nodes = zeros(1,NumNodes);
for j =1:length(Col3)
    [Row4,Col4]= find(Nd_Lk_Mtx(Col3(j),:));
    if length(Col4)>=1
        Merg_Nodes(1,Col3(j))=1;
    end
end
[Row5,Col5]=ind2sub(size(Merg_Nodes),find(Merg_Nodes(1,:)==1));
All_Coding_Node = Col5;

%% Find Parent chiled nodes
Parent_Chiled = {};
%All_Paths={};
for i=1:NumNodes
    for j=1:NumNodes
        Parent_Chiled{1,i}(1,j)= Nd_Lk_Mtx(i,j);
        Parent_Chiled{2,i}=find(Parent_Chiled{1,i});
        Parent_Chiled{3,i}=size(Parent_Chiled{2,i});
    end
end
```

```
toc  %Measure performance using stopwatch timer
```

4 MATLAB CODES FOR IMPLEMENTING THE AUGMENTING PATH ALGORITHM

```
%%
*********************************************************************
****
 %  Implementation Augmenting Path Algorithm

 %  Lalith P. Karunarathne, BEng(Hons)
 %  University Of Warwick, Coventry
*********************************************************************
****

%% All Paths Identification - Maximum 6 Hope Only

tic  %Measure performance using stopwatch timer

All_Paths={}; All_Paths1={}; All_Paths2={};All_Paths3={};
All_Paths4={};
All_Paths5={}; All_Paths6={};

Final_All_Paths1={}; Final_All_Paths2={}; Final_All_Paths3={};
Final_All_Paths4={}; Final_All_Paths5={}; Final_All_Paths6={};

Par_Chi_All_Paths={}; Par_Chi_All_Paths1={}; Par_Chi_All_Paths2={};
Par_Chi_All_Paths3={}; Par_Chi_All_Paths4={}; Par_Chi_All_Paths5={};

size_All_Paths1={}; size_Par_Chi_All_Paths1={};
size_Par_Chi_All_Paths2={};
size_Par_Chi_All_Paths3={};size_Par_Chi_All_Paths4={};size_Par_Chi_A
ll_Paths5={};


for k=1:length(All_NeWok_Sources)
    All_Paths{1,k}(1,1)= All_NeWok_Sources(k);
    % Hope 1
    for i=1:NumNodes
        Par_Chi_All_Paths{1,k}=
Parent_Chiled{2,All_Paths{1,k}(1,1)};
    end
    size_Par_Chi_All_Paths{1,k}=size(Par_Chi_All_Paths{1,k});
    for t=1:size_Par_Chi_All_Paths{1,k}(1,2)
        All_Paths1{k,t}(1,1)= All_Paths{1,k}(1,1);
        All_Paths1{k,t}(1,2)= Par_Chi_All_Paths{1,k}(1,t);

        % Test for any completed paths - Hope 1
        Test_One_zero1{k,t}=zeros(3,NumNodes);
        for j=1:length(All_NeWok_Sinks)
            Test_One_zero1{k,t}(1,All_NeWok_Sinks(j))=1;
        end
        Test_One_zero1{k,t}(2,All_Paths1{k,t}(1,2))=1;

        for i=1:NumNodes
            if Test_One_zero1{k,t}(1,i)==1 &&
Test_One_zero1{k,t}(2,i)==1
                Test_One_zero1{k,t}(3,i)=1;
            end
            All_Paths1{k,t}(2,i)= Test_One_zero1{k,t}(3,i);

            if All_Paths1{k,t}(2,i)~=0
                Final_All_Paths1{k,t}(1,[1 2])=All_Paths1{k,t}(1,[1
2]);
            end
        end

        %***** Hope 2
        for i=1:NumNodes
            Par_Chi_All_Paths1{k,t}=
Parent_Chiled{2,All_Paths1{k,t}(1,2)};
        end

        size_Par_Chi_All_Paths1{k,t}=size(Par_Chi_All_Paths1{k,t});
        for t1=1:size_Par_Chi_All_Paths1{k,t}(1,2)
            All_Paths2{k,t,t1}(1,[1 2])= All_Paths1{k,t}(1,[1 2]);
            All_Paths2{k,t,t1}(1,3)=Par_Chi_All_Paths1{k,t}(1,t1);

            %Test for any completed paths - Hope 2
            Test_One_zero2{k,t,t1}=zeros(3,NumNodes);
            for j=1:length(All_NeWok_Sinks)
                Test_One_zero2{k,t,t1}(1,All_NeWok_Sinks(j))=1;
            end
            Test_One_zero2{k,t,t1}(2,All_Paths2{k,t,t1}(1,3))=1;

            for i=1:NumNodes
                if Test_One_zero2{k,t,t1}(1,i)==1 &&
Test_One_zero2{k,t,t1}(2,i)==1
                    Test_One_zero2{k,t,t1}(3,i)=1;
                end
                All_Paths2{k,t,t1}(2,i)=
Test_One_zero2{k,t,t1}(3,i);
                if All_Paths2{k,t,t1}(2,i)~=0
                    Final_All_Paths2{k,t,t1}(1,[1 2
3])=All_Paths2{k,t,t1}(1,[1 2 3]);
                end
            end
            % ******** Hope 3
            for i=1:NumNodes
                Par_Chi_All_Paths2{k,t,t1}=
Parent_Chiled{2,All_Paths2{k,t,t1}(1,3)};
            end

size_Par_Chi_All_Paths2{k,t,t1}=size(Par_Chi_All_Paths2{k,t,t1});

            for t2=1:size_Par_Chi_All_Paths2{k,t,t1}(1,2)
                All_Paths3{k,t,t1,t2}(1,[1 2 3])=
All_Paths2{k,t,t1}(1,[1 2 3]);

All_Paths3{k,t,t1,t2}(1,4)=Par_Chi_All_Paths2{k,t,t1}(1,t2);
```

```
                %Test for any completed paths - Hope 3
                Test_One_zero3{k,t,t1,t2}=zeros(3,NumNodes);
                for j=1:length(All_NeWok_Sinks)

Test_One_zero3{k,t,t1,t2}(1,All_NeWok_Sinks(j))=1;
                end

Test_One_zero3{k,t,t1,t2}(2,All_Paths3{k,t,t1,t2}(1,4))=1;

                for i=1:NumNodes
                    if Test_One_zero3{k,t,t1,t2}(1,i)==1 &&
Test_One_zero3{k,t,t1,t2}(2,i)==1
                        Test_One_zero3{k,t,t1,t2}(3,i)=1;
                    end
                    All_Paths3{k,t,t1,t2}(2,i)=
Test_One_zero3{k,t,t1,t2}(3,i);
                    if All_Paths3{k,t,t1,t2}(2,i)~=0
                        Final_All_Paths3{k,t,t1,t2}(1,[1 2 3
4])=All_Paths3{k,t,t1,t2}(1,[1 2 3 4]);
                    end
                end

                % ******** Hope 4
                for i=1:NumNodes
                    Par_Chi_All_Paths3{k,t,t1,t2}=
Parent_Chiled{2,All_Paths3{k,t,t1,t2}(1,4)};
                end

size_Par_Chi_All_Paths3{k,t,t1,t2}=size(Par_Chi_All_Paths3{k,t,t1,t2
});

                for t3=1:size_Par_Chi_All_Paths3{k,t,t1,t2}(1,2)
                    All_Paths4{k,t,t1,t2,t3}(1,[1 2 3 4])=
All_Paths3{k,t,t1,t2}(1,[1 2 3 4]);

All_Paths4{k,t,t1,t2,t3}(1,5)=Par_Chi_All_Paths3{k,t,t1,t2}(1,t3);

                    %Test for any completed paths - Hope 4
                    Test_One_zero4{k,t,t1,t2,t3}=zeros(3,NumNodes);
                    for j=1:length(All_NeWok_Sinks)

Test_One_zero4{k,t,t1,t2,t3}(1,All_NeWok_Sinks(j))=1;
                    end

Test_One_zero4{k,t,t1,t2,t3}(2,All_Paths4{k,t,t1,t2,t3}(1,5))=1;

                    for i=1:NumNodes
                        if Test_One_zero4{k,t,t1,t2,t3}(1,i)==1 &&
Test_One_zero4{k,t,t1,t2,t3}(2,i)==1
                            Test_One_zero4{k,t,t1,t2,t3}(3,i)=1;
                        end
                        All_Paths4{k,t,t1,t2,t3}(2,i)=
Test_One_zero4{k,t,t1,t2,t3}(3,i);
                        if All_Paths4{k,t,t1,t2,t3}(2,i)~=0
                            Final_All_Paths4{k,t,t1,t2,t3}(1,[1 2 3
4 5])=All_Paths4{k,t,t1,t2,t3}(1,[1 2 3 4 5]);
                        end
                    end

                    % ******** Hope 5
                    for i=1:NumNodes
                        Par_Chi_All_Paths4{k,t,t1,t2,t3}=
Parent_Chiled{2,All_Paths4{k,t,t1,t2,t3}(1,5)};
                    end

size_Par_Chi_All_Paths4{k,t,t1,t2,t3}=size(Par_Chi_All_Paths4{k,t,t1
,t2,t3});

                    for
t4=1:size_Par_Chi_All_Paths4{k,t,t1,t2,t3}(1,2)
                        All_Paths5{k,t,t1,t2,t3,t4}(1,[1 2 3 4 5])=
All_Paths4{k,t,t1,t2,t3}(1,[1 2 3 4 5]);

All_Paths5{k,t,t1,t2,t3,t4}(1,6)=Par_Chi_All_Paths4{k,t,t1,t2,t3}(1,
t4);

                        %Test for any completed paths - Hope 5

Test_One_zero5{k,t,t1,t2,t3,t4}=zeros(3,NumNodes);
                        for j=1:length(All_NeWok_Sinks)

Test_One_zero5{k,t,t1,t2,t3,t4}(1,All_NeWok_Sinks(j))=1;
                        end

Test_One_zero5{k,t,t1,t2,t3,t4}(2,All_Paths5{k,t,t1,t2,t3,t4}(1,6))=
1;

                        for i=1:NumNodes
                            if
Test_One_zero5{k,t,t1,t2,t3,t4}(1,i)==1 &&
Test_One_zero5{k,t,t1,t2,t3,t4}(2,i)==1

Test_One_zero5{k,t,t1,t2,t3,t4}(3,i)=1;
                            end
                            All_Paths5{k,t,t1,t2,t3,t4}(2,i)=
Test_One_zero5{k,t,t1,t2,t3,t4}(3,i);
                            if All_Paths5{k,t,t1,t2,t3,t4}(2,i)~=0

Final_All_Paths5{k,t,t1,t2,t3,t4}(1,[1 2 3 4 5
6])=All_Paths5{k,t,t1,t2,t3,t4}(1,[1 2 3 4 5 6]);
                            end
                        end

                        % ******** Hope 6
                        for i=1:NumNodes
                            Par_Chi_All_Paths5{k,t,t1,t2,t3,t4}=
Parent_Chiled{2,All_Paths5{k,t,t1,t2,t3,t4}(1,6)};
                        end

size_Par_Chi_All_Paths5{k,t,t1,t2,t3,t4}=size(Par_Chi_All_Paths5{k,t
,t1,t2,t3,t4});

                        for
t5=1:size_Par_Chi_All_Paths5{k,t,t1,t2,t3,t4}(1,2)
```

```
                            All_Paths6{k,t,t1,t2,t3,t4,t5}(1,[1 2 3
4 5 6])= All_Paths5{k,t,t1,t2,t3,t4}(1,[1 2 3 4 5 6]);

All_Paths6{k,t,t1,t2,t3,t4,t5}(1,7)=Par_Chi_All_Paths5{k,t,t1,t2,t3,
t4}(1,t5);

                            %Test for any completed paths - Hope 6

Test_One_zero6{k,t,t1,t2,t3,t4,t5}=zeros(3,NumNodes);
                            for j=1:length(All_NeWok_Sinks)

Test_One_zero6{k,t,t1,t2,t3,t4,t5}(1,All_NeWok_Sinks(j))=1;
                            end

Test_One_zero6{k,t,t1,t2,t3,t4,t5}(2,All_Paths6{k,t,t1,t2,t3,t4,t5}(
1,7))=1;
                            for i=1:NumNodes
                                if
Test_One_zero6{k,t,t1,t2,t3,t4,t5}(1,i)==1 &&
Test_One_zero6{k,t,t1,t2,t3,t4,t5}(2,i)==1

Test_One_zero6{k,t,t1,t2,t3,t4,t5}(3,i)=1;
                                end
                                All_Paths6{k,t,t1,t2,t3,t4,t5}(2,i)=
Test_One_zero6{k,t,t1,t2,t3,t4,t5}(3,i);
                                if
All_Paths6{k,t,t1,t2,t3,t4,t5}(2,i)~=0

Final_All_Paths6{k,t,t1,t2,t3,t4,t5}(1,[1 2 3 4 5 6
7])=All_Paths6{k,t,t1,t2,t3,t4,t5}(1,[1 2 3 4 5 6 7]);
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
 toc %Measure performance using stopwatch timer
```

5 MATLAB CODES FOR IMPLEMENTING THE LINEAR DISJOINT PATH ALGORITHM

```
%%
********************************************************************
****
 %  Implementation: Linear Disjoint Paths Algorithm

 %  Lalith P. Karunarathne, BEng(Hons)
 %  University Of Warwick, Coventry
********************************************************************
****

%% Linear Disjoint Paths Algorithm
 % It Should be modified when add more sources
```

```
tic %Measure performance using stopwatch timer
final_Path_from_S1_to_Sk={};
Test_fi_fro_S1_to_Sk = {};
for k=1:length(All_NeWok_Sources)
    final_Path_from_S1_to_Sk{k,1}=(cat(1,Final_All_Paths1{k,:}));
    final_Path_from_S1_to_Sk{k,2}=(cat(1,Final_All_Paths2{k,:,:}));

final_Path_from_S1_to_Sk{k,3}=(cat(1,Final_All_Paths3{k,:,:,:}));

final_Path_from_S1_to_Sk{k,4}=(cat(1,Final_All_Paths4{k,:,:,:,:}));

final_Path_from_S1_to_Sk{k,5}=(cat(1,Final_All_Paths5{k,:,:,:,:,:}))
;

final_Path_from_S1_to_Sk{k,6}=(cat(1,Final_All_Paths6{k,:,:,:,:,:,:}
));

    for i=1:6

Test_fi_fro_S1_to_Sk{k,i}=size(final_Path_from_S1_to_Sk{k,i});
    end
end

%% Based on Source S1
S1_All_Paths1={};S1_All_Paths2={};S1_All_Paths3={};S1_All_Paths4={};
S1_All_Paths5={};S1_All_Paths6={};

for i=1:Test_fi_fro_S1_to_Sk{1,1}(1,1)
    S1_All_Paths1{1,i}(1,NumNodes)=0;
    S1_All_Paths1{1,i}=final_Path_from_S1_to_Sk{1,1}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{1,2}(1,1)
    S1_All_Paths2{1,i}(1,NumNodes)=0;
    S1_All_Paths2{1,i}=final_Path_from_S1_to_Sk{1,2}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{1,3}(1,1)
    S1_All_Paths3{1,i}(1,NumNodes)=0;
    S1_All_Paths3{1,i}=final_Path_from_S1_to_Sk{1,3}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{1,4}(1,1)
    S1_All_Paths4{1,i}(1,NumNodes)=0;
    S1_All_Paths4{1,i}=final_Path_from_S1_to_Sk{1,4}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{1,5}(1,1)
    S1_All_Paths5{1,i}(1,NumNodes)=0;
    S1_All_Paths5{1,i}=final_Path_from_S1_to_Sk{1,5}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{1,6}(1,1)
    S1_All_Paths6{1,i}(1,NumNodes)=0;
    S1_All_Paths6{1,i}=final_Path_from_S1_to_Sk{1,6}(i,:);
end



%% Based on Source S2
S2_All_Paths1={};S2_All_Paths2={};S2_All_Paths3={};S2_All_Paths4={};
S2_All_Paths5={};S2_All_Paths6={};
for i=1:Test_fi_fro_S1_to_Sk{2,1}(1,1)
```

```
    S2_All_Paths1{1,i}(1,NumNodes)=0;
    S2_All_Paths1{1,i}=final_Path_from_S1_to_Sk{2,1}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{2,2}(1,1)
    S2_All_Paths2{1,i}(1,NumNodes)=0;
    S2_All_Paths2{1,i}=final_Path_from_S1_to_Sk{2,2}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{2,3}(1,1)
    S2_All_Paths3{1,i}(1,NumNodes)=0;
    S2_All_Paths3{1,i}=final_Path_from_S1_to_Sk{2,3}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{2,4}(1,1)
    S2_All_Paths4{1,i}(1,NumNodes)=0;
    S2_All_Paths4{1,i}=final_Path_from_S1_to_Sk{2,4}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{2,5}(1,1)
    S2_All_Paths5{1,i}(1,NumNodes)=0;
    S2_All_Paths5{1,i}=final_Path_from_S1_to_Sk{2,5}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{2,6}(1,1)
    S2_All_Paths6{1,i}(1,NumNodes)=0;
    S2_All_Paths6{1,i}=final_Path_from_S1_to_Sk{2,6}(i,:);
end

%% Based on Source S3
S3_All_Paths1={};S3_All_Paths2={};S3_All_Paths3={};S3_All_Paths4={};
S3_All_Paths5={};S3_All_Paths6={};
for i=1:Test_fi_fro_S1_to_Sk{3,1}(1,1)
    S3_All_Paths1{1,i}(1,NumNodes)=0;
    S3_All_Paths1{1,i}=final_Path_from_S1_to_Sk{3,1}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{3,2}(1,1)
    S3_All_Paths2{1,i}(1,NumNodes)=0;
    S3_All_Paths2{1,i}=final_Path_from_S1_to_Sk{3,2}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{3,3}(1,1)
    S3_All_Paths3{1,i}(1,NumNodes)=0;
    S3_All_Paths3{1,i}=final_Path_from_S1_to_Sk{3,3}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{3,4}(1,1)
    S3_All_Paths4{1,i}(1,NumNodes)=0;
    S3_All_Paths4{1,i}=final_Path_from_S1_to_Sk{3,4}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{3,5}(1,1)
    S3_All_Paths5{1,i}(1,NumNodes)=0;
    S3_All_Paths5{1,i}=final_Path_from_S1_to_Sk{3,5}(i,:);
end
for i=1:Test_fi_fro_S1_to_Sk{3,6}(1,1)
    S3_All_Paths6{1,i}(1,NumNodes)=0;
    S3_All_Paths6{1,i}=final_Path_from_S1_to_Sk{3,6}(i,:);
end
%%
Cat_S1_All_Paths=horzcat(S1_All_Paths1,S1_All_Paths2,S1_All_Paths3,S
1_All_Paths4,S1_All_Paths5,S1_All_Paths6);
Cat_S2_All_Paths=horzcat(S2_All_Paths1,S2_All_Paths2,S2_All_Paths3,S
2_All_Paths4,S2_All_Paths5,S2_All_Paths6);
Cat_S3_All_Paths=horzcat(S3_All_Paths1,S3_All_Paths2,S3_All_Paths3,S
3_All_Paths4,S3_All_Paths5,S3_All_Paths6);
%%
```

```
size_Cat_S1_All=size(Cat_S1_All_Paths);
Spar_Cat_S1_All_Paths = {};
size_Cat_S1_All_Paths={};
for i=1:size_Cat_S1_All(1,2)
    size_Cat_S1_All_Paths{1,i}=size(Cat_S1_All_Paths{1,i});
    for i1=1:NumNodes
        for j=1:size_Cat_S1_All_Paths{1,i}(1,2)
            Spar_Cat_S1_All_Paths{1,i}(1,i1)=0;

Spar_Cat_S1_All_Paths{1,i}(1,j)=Cat_S1_All_Paths{1,i}(1,j);

Spar_Cat_S1_All_Paths{1,i}(2,Cat_S1_All_Paths{1,i}(1,j))=1;
        end
    end
end
%%
size_Cat_S2_All=size(Cat_S2_All_Paths);
Spar_Cat_S2_All_Paths = {};
size_Cat_S2_All_Paths= {};
for i=1:size_Cat_S2_All(1,2)
    size_Cat_S2_All_Paths{1,i}=size(Cat_S2_All_Paths{1,i});
    for i1=1:NumNodes
        for j=1:size_Cat_S2_All_Paths{1,i}(1,2)
            Spar_Cat_S2_All_Paths{1,i}(1,i1)=0;

Spar_Cat_S2_All_Paths{1,i}(1,j)=Cat_S2_All_Paths{1,i}(1,j);

Spar_Cat_S2_All_Paths{1,i}(2,Cat_S2_All_Paths{1,i}(1,j))=1;
        end
    end
end
%%
size_Cat_S3_All=size(Cat_S3_All_Paths);
Spar_Cat_S3_All_Paths = {};
size_Cat_S3_All_Paths = {};
for i=1:size_Cat_S3_All(1,2)
    size_Cat_S3_All_Paths{1,i}=size(Cat_S3_All_Paths{1,i});
    for i1=1:NumNodes
        for j=1:size_Cat_S3_All_Paths{1,i}(1,2)
            Spar_Cat_S3_All_Paths{1,i}(1,i1)=0;

Spar_Cat_S3_All_Paths{1,i}(1,j)=Cat_S3_All_Paths{1,i}(1,j);

Spar_Cat_S3_All_Paths{1,i}(2,Cat_S3_All_Paths{1,i}(1,j))=1;
        end
    end
end
%%
Comb_Cat_S1_S2_All={};
for i=1:size_Cat_S1_All(1,2)
    for j=1:size_Cat_S2_All(1,2)

Comb_Cat_S1_S2_All{i,j}(1,:)=Spar_Cat_S1_All_Paths{1,i}(1,:);

Comb_Cat_S1_S2_All{i,j}(2,:)=Spar_Cat_S2_All_Paths{1,j}(1,:);

Comb_Cat_S1_S2_All{i,j}(3,:)=Spar_Cat_S1_All_Paths{1,i}(2,:);

Comb_Cat_S1_S2_All{i,j}(4,:)=Spar_Cat_S2_All_Paths{1,j}(2,:);
```

```
        Comb_Cat_S1_S2_All{i,j}(5,:)=Test_All_Net_Sink(1,:);
    end
end

%%
for i=1:size_Cat_S1_All(1,2)
    for j=1:size_Cat_S2_All(1,2)
        for j1=1:NumNodes
            if Comb_Cat_S1_S2_All{i,j}(3,j1)==1 &&
Comb_Cat_S1_S2_All{i,j}(4,j1)==1 && Comb_Cat_S1_S2_All{i,j}(5,j1)==1
                Comb_Cat_S1_S2_All{i,j}(6,j1)=1;
            end
        end
    end
end
%%
size_Comb_Cat_S1_S2_All={};
for i=1:size_Cat_S1_All(1,2)
    for j=1:size_Cat_S2_All(1,2)
        Test_One_Zero_S1_S2(i,j)=0;
        size_Comb_Cat_S1_S2_All{i,j}=size(Comb_Cat_S1_S2_All{i,j});
        if size_Comb_Cat_S1_S2_All{i,j}(1,1)==6
            Test_One_Zero_S1_S2(i,j)=1;
        end
    end
end
[Row1,Col1]=ind2sub(size(Test_One_Zero_S1_S2),find(Test_One_Zero_S1_
S2(:,:)==1));
sizRow1=size(Row1);
%%
Lineup_Comb_Cat_S1_S2={};
for i=1:sizRow1(1,1) %length(Row1)
    Lineup_Comb_Cat_S1_S2{1,i}= Comb_Cat_S1_S2_All{Row1(i),Col1(i)};
End

%Comb_Cat_S1_S2_All{1,1}
%Lineup_Comb_Cat_S1_S2{1,1}
%%
size_Lineup_Comb_Cat_S1_S2=size(Lineup_Comb_Cat_S1_S2);
Comb_Cat_S1_S2_S3_All={};
for i=1:size_Lineup_Comb_Cat_S1_S2(1,2)
    for j=1:size_Cat_S3_All(1,2)

        Comb_Cat_S1_S2_S3_All{i,j}=Lineup_Comb_Cat_S1_S2{1,i}([1
2],:);

Comb_Cat_S1_S2_S3_All{i,j}(3,:)=Spar_Cat_S3_All_Paths{1,j}(1,:);

Comb_Cat_S1_S2_S3_All{i,j}(4,:)=Lineup_Comb_Cat_S1_S2{1,i}(3,:);

Comb_Cat_S1_S2_S3_All{i,j}(5,:)=Lineup_Comb_Cat_S1_S2{1,i}(4,:);

Comb_Cat_S1_S2_S3_All{i,j}(6,:)=Spar_Cat_S3_All_Paths{1,j}(2,:);

Comb_Cat_S1_S2_S3_All{i,j}(7,:)=Lineup_Comb_Cat_S1_S2{1,i}(6,:);
    end
end
%%
for i=1:size_Lineup_Comb_Cat_S1_S2(1,2)
```

```
    for j=1:size_Cat_S3_All(1,2)
        for j1=1:NumNodes
            if Comb_Cat_S1_S2_S3_All{i,j}(6,j1)==1 &&
Comb_Cat_S1_S2_S3_All{i,j}(7,j1)==1
                Comb_Cat_S1_S2_S3_All{i,j}(8,j1)=1;
            end
        end
    end
end

%%
size_Comb_Cat_S1_S2_S3_All={};
for i=1:size_Lineup_Comb_Cat_S1_S2(1,2)
    for j=1:size_Cat_S3_All(1,2)
        Test_One_Zero_S1_S2_S3(i,j)=0;

size_Comb_Cat_S1_S2_S3_All{i,j}=size(Comb_Cat_S1_S2_S3_All{i,j});
        if size_Comb_Cat_S1_S2_S3_All{i,j}(1,1)==8
            Test_One_Zero_S1_S2_S3(i,j)=1;
        end

    end
end
[Row2,Col2]=ind2sub(size(Test_One_Zero_S1_S2_S3),find(Test_One_Zero_
S1_S2_S3(:,:)==1));
%%
Lineup_Comb_Cat_S1_S2_S3={};
for i=1:length(Row2)
    Lineup_Comb_Cat_S1_S2_S3{1,i}=
Comb_Cat_S1_S2_S3_All{Row2(i),Col2(i)};
end
%% Identify the sets of the Linear disjoint paths
siz_Li_up_Cmb_Cat_S1_S2_S3=size(Lineup_Comb_Cat_S1_S2_S3);
Test_One_Zero_Lin_S1_S2_S3={};
for i=1:siz_Li_up_Cmb_Cat_S1_S2_S3(1,2)
    for j=1:NumNodes
        Test_One_Zero_Lin_S1_S2_S3{1,i}(1,j)=0;
        if Lineup_Comb_Cat_S1_S2_S3{1,i}(4,j)==1 &&
Lineup_Comb_Cat_S1_S2_S3{1,i}(5,j)==1 ||...
                Lineup_Comb_Cat_S1_S2_S3{1,i}(4,j)==1 &&
Lineup_Comb_Cat_S1_S2_S3{1,i}(6,j)==1 ||...
                Lineup_Comb_Cat_S1_S2_S3{1,i}(5,j)==1 &&
Lineup_Comb_Cat_S1_S2_S3{1,i}(6,j)==1 ||...
                Lineup_Comb_Cat_S1_S2_S3{1,i}(4,j)==1 &&
Lineup_Comb_Cat_S1_S2_S3{1,i}(5,j)==1 &&...
                Lineup_Comb_Cat_S1_S2_S3{1,i}(6,j)==1
            Test_One_Zero_Lin_S1_S2_S3{1,i}(1,j)=1;
        end

    end
end
%% Feasible set of Lineat Disjoint paths
find_Test_one_zero_S1_S2_S3={};
for i=1:siz_Li_up_Cmb_Cat_S1_S2_S3(1,2)
    Test_find_one_zero_S1_S2_S3(1,i)=0;
    find_Test_one_zero_S1_S2_S3{1,i}=
find(Test_One_Zero_Lin_S1_S2_S3{1,i});
    find_Test_one_zero_S1_S2_S3{1,i}(2,[1
2])=size(find_Test_one_zero_S1_S2_S3{1,i});
```

```
    if find_Test_one_zero_S1_S2_S3{1,i}(2,2)==1
        Test_find_one_zero_S1_S2_S3(1,i)=1;
    end

end

[Row3,Col3]=ind2sub(size(Test_find_one_zero_S1_S2_S3),find(Test_find
_one_zero_S1_S2_S3(:,:)==1));
%% Line up Set of Linear Disjoint path sets
Lineup_All_Linr_Disjo_Path ={};
for i=1:length(Col3)

Lineup_All_Linr_Disjo_Path{1,i}=Lineup_Comb_Cat_S1_S2_S3{1,Col3(i)};
end
%%
size_Lin_All_Linr_Disjo_Path=size(Lineup_All_Linr_Disjo_Path);
Comb_Sink_Set_Lin_Dis={};
for i=1:length(All_NeWok_Sinks)
    for j=1:size_Lin_All_Linr_Disjo_Path(1,2)
        Comb_Sink_Set_Lin_Dis{i,j}([1 2 3 4],:)=
Lineup_All_Linr_Disjo_Path{1,j}([1 2 3 7],:);
        for j1=1:NumNodes
            Comb_Sink_Set_Lin_Dis{i,j}(5,All_NeWok_Sinks(i))=1;
        end
    end
end
%% Comparison
Test_Comb_Sink_Set_Lin_Dis={};
find_Test_Comb_Sink_Set_Lin_Dis={};
for i=1:length(All_NeWok_Sinks)
    for j=1:size_Lin_All_Linr_Disjo_Path(1,2)
        Test_One_Zero_Comb_Sink_Set_Lin(i,j)=0;
        for j1=1:NumNodes
            if Comb_Sink_Set_Lin_Dis{i,j}(4,j1)==1 &&
Comb_Sink_Set_Lin_Dis{i,j}(5,j1)==1
                Test_One_Zero_Comb_Sink_Set_Lin(i,j)=1;

            end

Test_Comb_Sink_Set_Lin_Dis{1,i}(1,j)=Test_One_Zero_Comb_Sink_Set_Lin
(i,j);

find_Test_Comb_Sink_Set_Lin_Dis{1,i}=find(Test_Comb_Sink_Set_Lin_Dis
{1,i});
        end
    end
end
%%
Size_fi_Tet__Lin_Dis=size(find_Test_Comb_Sink_Set_Lin_Dis);
size_fin_Te_Comb_Sink_Lin_Dis={};
Set_Linr_Disjo_base_Sink={};
for i=1:Size_fi_Tet__Lin_Dis(1,2)

size_fin_Te_Comb_Sink_Lin_Dis{1,i}=size(find_Test_Comb_Sink_Set_Lin_
Dis{1,i});
    for j=1:size_fin_Te_Comb_Sink_Lin_Dis{1,i}(1,2)
```

```
Set_Linr_Disjo_base_Sink{j,i}=Comb_Sink_Set_Lin_Dis{i,find_Test_Comb
_Sink_Set_Lin_Dis{1,i}(1,j)}([1 2 3],:);
    end
end
toc %Measure performance using stopwatch timer
```

6 MATLAB CODES FOR SELECT THE INITIAL POPULATION

```
%% Implementation: select the Initial Population

 %  Lalith P. Karunarathne, BEng(Hons)
 %  University Of Warwick, Coventry
% ****************************************************
% Randomly select the initial population

tic  %Measure performance using stopwatch timer

Population_Size=100;
value_Set={};
Perm_Matrix=[];
for i=1:Size_fi_Tet__Lin_Dis(1,2)
    for j=1:Population_Size

value_Set{j,i}=randperm(size_fin_Te_Comb_Sink_Lin_Dis{1,i}(1,2));
        Perm_Matrix(j,i)=value_Set{j,i}(1,1);
    end
end
Population={};
for i=1:Size_fi_Tet__Lin_Dis(1,2)
    for j=1:Population_Size

Population{j,i}=Set_Linr_Disjo_base_Sink{Perm_Matrix(j,i),i};
    end
end

toc %Measure performance using stopwatch timer
```

7 MATLAB CODES FOR IMPLEMENT THE CROSSOVER AND MUTATION OPERATORS IN MULTI OBJECTIVE – GA (MOGA)

```
%%*****************************************************************
********
%% Implementation: Multi - Objective GA
%  Lalith P. Karunarathne, BEng(Hons)
%  University Of Warwick, Coventry
%*************************************************

%% Multi - Objective GA (Crossover & Mutation operation)
```

```
tic %Measure performance using stopwatch timer
siz_Popula=size(Population);
NubOfGen=2;
Crosoverd_Popu={};
Previous_Genran={};
NewGeneration={};
CrossoverNewGen={};
Muta_Crosoverd_Popu={};

for ig =1:NubOfGen

    if ig==1
        Previous_Genran=Population;


run('C:\Users\LALITHK\Documents\MATLAB\First_ReseEvolve\GFitnessEvol
ve')

    else
        %Crossover>>>>

        for ip =1:siz_Popula(1,1)
            for jp = 1:siz_Popula(1,2)
                if ip <= (siz_Popula(1,1)/2)
                Crosoverd_Popu{2*ip-1,jp}= NewGeneration{(2*ip),jp};
                Crosoverd_Popu{2*ip-1,5}= NewGeneration{(2*ip-1),5};
                Crosoverd_Popu{2*ip-1,6}= NewGeneration{(2*ip-1),6};

                Crosoverd_Popu{2*ip,jp}= Crosoverd_Popu{(2*ip-
1),jp};
                Crosoverd_Popu{2*ip,5}= Crosoverd_Popu{(2*ip),5};
                Crosoverd_Popu{2*ip,6}= Crosoverd_Popu{(2*ip),6};
                end

            end
        end

      %Mutation>>>>>
      for ip =1:siz_Popula(1,1)*2
            for jp = 1:siz_Popula(1,2)
            Muta_Crosoverd_Popu{ip,jp}= Crosoverd_Popu{ip,jp};

            ra_Pernum = randperm(siz_Popula(1,2));
            ra_Pernumx = randperm(siz_Popula(1,1));
            ra_ Pernumy = randperm(length(All_NeWok_Sources));

      Muta_Crosoverd_Popu{ip, ra_Pernum }( ra_ Pernumy,:)= ...
      Crosoverd_Popu{ra_Pernumx, ra_Pernum}( ra_ Pernumy,:);

            end
      end

        if ig >1
            Previous_Genran= Muta_Crosoverd_Popu;

run('C:\Users\LALITHK\Documents\MATLAB\First_ReseEvolve\GFitnessEvol
ve')
```

```
        end
    end

end
```

8 MATLAB CODES FOR IMPLEMENT THE SELECTOR OPERATOR, FITNESS ASSIGNMENT AND INDIVIDUAL EVALUATION FOR MULTI – OBJECTIVE GA (MOGA)

```
%*******************************************************************
*****
%% Implementation: Fitness assignment and individual evaluation for
Multi - Objective GA
%  Lalith P. Karunarathne, BEng(Hons)
%  University Of Warwick, Coventry
%*******************************************************************
***

%% Fitness Evoluton for Generation
%Previous_Genran

Num_Source=length(All_NeWok_Sources);
Trans_Population_fist={};
Trans_Pop_link_fist={};
Dub_Num_Source=2*Num_Source;

for i=1:Size_fi_Tet__Lin_Dis(1,2) %<-Number of sinks
    for j=1:Population_Size
        for i1=1:NumNodes
            for j1=1:Num_Source


Trans_Population_fist{j,i}(i1,j1)=Previous_Genran{j,i}(j1,i1);
            end
        end
    end
end
odd_Num=(1:2:Dub_Num_Source);
even_Num=(2:2:Dub_Num_Source);
Row_Index=(2:NumNodes);
Fina_Trans_Pop_link_fist={};

for i=1:Size_fi_Tet__Lin_Dis(1,2)
    for j=1:Population_Size
        for i1=1:length(Row_Index)
            for j1=1:length(odd_Num)
                Trans_Pop_link_fist{j,i}(i1,odd_Num(j1))=
Trans_Population_fist{j,i}(i1,j1);
                Trans_Pop_link_fist{j,i}(i1,even_Num(j1))=
Trans_Population_fist{j,i}(Row_Index(i1),j1);
                if Trans_Pop_link_fist{j,i}(i1,odd_Num(j1))~=0 &&
Trans_Pop_link_fist{j,i}(i1,even_Num(j1))~=0
                    Fina_Trans_Pop_link_fist{j,i}(i1,odd_Num(j1))=
Trans_Pop_link_fist{j,i}(i1,odd_Num(j1));
```

```
                    Fina_Trans_Pop_link_fist{j,i}(i1,even_Num(j1))=
Trans_Pop_link_fist{j,i}(i1,even_Num(j1));
                end
            end
        end
    end
end
%Previous_Genran{1,1}
%NewGeneration=Previous_Genran;
%%  Fina_Trans_Pop_link_fist{j,i} convert to spares matrix
% Fitness assignment
Vercat_Fina_Trans_Pop_link_fist={};
One_zero_Tran_Pop_fist={};
size_Fina_Trans_Pop_link_fist = size(Fina_Trans_Pop_link_fist);
size_Vercat_Fina_Trans_fist={};
Numof_links_individ=[];
Weight_Indivi={};
Weightof_links_individ=[];

for i=1:size_Fina_Trans_Pop_link_fist(1,1)
    Vercat_Fina_Trans_Pop_link_fist{i,1}=
vertcat(Fina_Trans_Pop_link_fist{i,:});

size_Vercat_Fina_Trans_fist{i,1}=size(Vercat_Fina_Trans_Pop_link_fis
t{i,1});
    for i1 =1:NumNodes
        for j1 =1:NumNodes
            One_zero_Tran_Pop_fist{i,1}(i1,j1)=0;
        end
    end
    for i2 =1:size_Vercat_Fina_Trans_fist{i,1}(1,1)
        for j2 =1:length(odd_Num)
            if
Vercat_Fina_Trans_Pop_link_fist{i,1}(i2,odd_Num(j2))~=0 &&
Vercat_Fina_Trans_Pop_link_fist{i,1}(i2,even_Num(j2))~=0

One_zero_Tran_Pop_fist{i,1}(Vercat_Fina_Trans_Pop_link_fist{i,1}(i2,
odd_Num(j2)),...

Vercat_Fina_Trans_Pop_link_fist{i,1}(i2,even_Num(j2)))=1;
            end
        end
    end
    Numof_links_individ(i,1)=sum(sum(One_zero_Tran_Pop_fist{i,1}));

    for i1 =1:NumNodes
        for j1 =1:NumNodes
            if One_zero_Tran_Pop_fist{i,1}(i1,j1)==1 &&
Weight_Mtx(i1,j1)~=0
                Weight_Indivi{i,1}(i1,j1)=Weight_Mtx(i1,j1);
            end
        end
    end
   Weightof_links_individ(i,1)=sum(sum(Weight_Indivi{i,1}));

end

min_Numof_links_indi=min(Numof_links_individ);
```

```
min_Weightof_links_ind=min(Weightof_links_individ);

%One_zero_Tran_Pop_fist{1,1}
%Weight_Indivi{1,1}
%Weightof_links_individ(1,1)
%Numof_links_individ(2,1)

One_Zero_Coding_Colm_fist={};
for i=1:size_Fina_Trans_Pop_link_fist(1,1)
    for i1 =1:NumNodes
        for j1 =1:NumNodes
            One_Zero_Coding_Colm_fist{i,1}(i1,j1)=0;
        end
    end
    for j2=1:length(All_Coding_Node)
        One_Zero_Coding_Colm_fist{i,1}(:,All_Coding_Node(j2))=
One_zero_Tran_Pop_fist{i,1}(:,All_Coding_Node(j2));
    end
end
%One_Zero_Coding_Colm_fist{1,1}
%% Individual evaluation
size_Rowx ={};
size_Rowy ={};
find_size_Rowx={};
Num_Inlink_Coin_pt={};
Codin_Pt={};
Num_Inlink_each_Indi={};
Test_One_zero_size_Rowx={};
Average_inlinks_per_CodingPt=[];
plot_X=[];
plot_Y=[];
Presen_Gen=[];

%Pareto Fronts
% Objective 1
ParetoFront_CodinPts=2; % Optimal Pareto for Objective 1

% Objective 2
ParetoFront_Numof_links=min_Numof_links_indi; % Optimal Pareto for
Objective 2

% Objective 3
ParetoFrontWeightof_links=min_Weightof_links_ind; % Optimal Pareto
for Objective 3

for i=1:size_Fina_Trans_Pop_link_fist(1,1)
    for j=1:NumNodes
        [Rowx,Colmx]=find(One_Zero_Coding_Colm_fist{i,1}(:,j));
        size_Rowx{i,1}(j,[1 2])=size(Rowx);
        if size_Rowx{i,1}(j,1)<2
            size_Rowx{i,1}(j,1)=0;
        end
        % coding points index
        Codin_Pt{i,1}=find(size_Rowx{i,1}(:,1));

        Test_One_zero_size_Rowx{i,1}(1,j)=size_Rowx{i,1}(j,1);
        if Test_One_zero_size_Rowx{i,1}(1,j)~=0
            Test_One_zero_size_Rowx{i,1}(1,j)=1;
        end

        [Rowy,Colmy]=find(size_Rowx{i,1}(:,1));
        size_Rowy{i,1}(1,[1 2])=size(Rowy);
    end

    plot_X(i,1)=size_Rowy{i,1}(1,1);              % Objective -1,
Number of coding point
    Presen_Gen(i,1)=size_Rowy{i,1}(1,1);



    % Good individual - objective 1
    if (size_Rowy{i,1}(1,1)-ParetoFront_CodinPts)>=0 &&
(size_Rowy{i,1}(1,1)-ParetoFront_CodinPts)<=1
        plot_X(i,1)=size_Rowy{i,1}(1,1); %-ParetoFront_CodinPts;

    else
        plot_X(i,1)=0;
    end

    % Bad individual - objective 1
    if  (size_Rowy{i,1}(1,1)-ParetoFront_CodinPts)>2
        Bad_plot_X(i,1)=size_Rowy{i,1}(1,1);
    else
        Bad_plot_X(i,1)=0;
    end

    % Good Individual - objective 2
    if (Numof_links_individ(i,1)-ParetoFront_Numof_links)>=0 &&...
            (Numof_links_individ(i,1)-ParetoFront_Numof_links)<=1
        plot_Y(i,1)=Numof_links_individ(i,1); %-
ParetoFront_Numof_links;
    else
        plot_Y(i,1)=0.0000;
    end

    % Bad Individual - objective 2
    if  (Numof_links_individ(i,1)-ParetoFront_Numof_links)>2
        Bad_plot_Y(i,1)=Numof_links_individ(i,1);
    else
        Bad_plot_Y(i,1)=0.000;
    end

     % Good Individual - objective 3
    if (Weightof_links_individ(i,1)-min_Weightof_links_ind)>=0 &&...
            (Weightof_links_individ(i,1)-min_Weightof_links_ind)<=1
        plot_Z(i,1)=Weightof_links_individ(i,1); %-
min_Weightof_links_ind;
    else
        plot_Z(i,1)=0.0000;
    end

    % Bad Individual - objective 3
    if  (Weightof_links_individ(i,1)-min_Weightof_links_ind)>2
        Bad_plot_Z(i,1)=Weightof_links_individ(i,1);
    else
        Bad_plot_Z(i,1)=0.000;
    end
end

%%
%Vercat_Fina_Trans_Pop_link{1,1}
%Test_One_zero_Codin_Pt{[1 2],1}

size_Codin_Pt=size(Codin_Pt);
Test_size_Codin_Pt={};
Test_One_zero_Codin_Pt={};
for i=1:size_Codin_Pt(1,1)
    for j=1:NumNodes
        Test_One_zero_Codin_Pt{i,1}(1,j)=0;
    end
    Test_size_Codin_Pt{i,1}(1,[1 2])=size(Codin_Pt{i,1});
    for i1=1:Test_size_Codin_Pt{i,1}(1,1)
        Test_One_zero_Codin_Pt{i,1}(1,Codin_Pt{i,1}(i1,1))=1;

    end
end

size_Previous_Genran=size(Previous_Genran);
size_Cell_Previous_Genran=size(Previous_Genran{1,1});
Test_MatrPrevious_Genran=[];
%VerCat_Gen1_Gen2{1,2}
TestPrevious_Genran={};
TestPrevious_Cells={};
conca_TestPrevious_Cells={};
count_conca_Test=[];
count_Test_One_zero_Codin=[];
Coding_Resou_Shar_ratio=[];



for i=1:size_Previous_Genran(1,1)
    for j=1:size_Previous_Genran(1,2)
        for i1=1:size_Cell_Previous_Genran(1,1)
            for j1=1:size_Cell_Previous_Genran(1,2)
                TestPrevious_Cells{i,j}(1,j1)=0;
                if Previous_Genran{i,j}(i1,j1)~=0

TestPrevious_Genran{i,j}(i1,Previous_Genran{i,j}(i1,j1))=1;
                end

TestPrevious_Genran{i,j}(4,j1)=Test_One_zero_Codin_Pt{i,1}(1,j1);
                if TestPrevious_Genran{i,j}(1,j1)==1 &&
TestPrevious_Genran{i,j}(4,j1)==1||...
                        TestPrevious_Genran{i,j}(2,j1)==1 &&
TestPrevious_Genran{i,j}(4,j1)==1||...
                        TestPrevious_Genran{i,j}(3,j1)==1 &&
TestPrevious_Genran{i,j}(4,j1)==1

                    TestPrevious_Cells{i,j}(1,j1)=1;

                end
                %TestPrevious_Cells{i,j}(1,j1) =
TestPrevious_Genran{i,j}(5,j1);

conca_TestPrevious_Cells{i,1}(j,j1)=TestPrevious_Cells{i,j}(1,j1);
```

```
count_conca_Test(i,1)=nnz(conca_TestPrevious_Cells{i,1});

count_Test_One_zero_Codin(i,1)=nnz(Test_One_zero_Codin_Pt{i,1});

            end
        end
    end
end

%% 3D plot for fitness diagram
for i=1:length(plot_Y)

    if plot_X(i,1)~=0 && plot_Y(i,1)~=0 && plot_Z(i,1)~=0
        subplot(3,2,1);plot3(plot_X(i),plot_Y(i),plot_Z(i,1),'b*');

    end

    if Bad_plot_X(i,1)~=0 && Bad_plot_Y(i,1)~=0 &&
Bad_plot_Z(i,1)~=0

subplot(3,2,2);plot3(Bad_plot_X(i),Bad_plot_Y(i),Bad_plot_Z(i,1),'bo
');

    end

    clear on
    grid on
    box on
    hold on

end

%%
Feasib_Individual =[];
for i=1:length(plot_Y)
    if plot_X(i,1)~=0 && plot_Y(i,1)~=0 && plot_Z(i,1)~=0
        Feasib_Individual(i,1)=1;
    else
        Feasib_Individual(i,1)=0;
    end
end

fin_Feasib_Individual=find(Feasib_Individual);
siz_fin_Feasib_Indiv=size(fin_Feasib_Individual);
if siz_fin_Feasib_Indiv(1,1)>=4
    fprintf('Feasible Individual  =',toc)
end

Testgood_indi=[];
Testbad_indi=[];
for i=1:length(plot_Y)
    if plot_X(i,1)~=0 && plot_Y(i,1)~=0 && plot_Z(i,1)~=0
        Testgood_indi(i,1)=1;
    else
        Testgood_indi(i,1)=0;
    end
    if Bad_plot_X(i,1)~=0 && Bad_plot_Y(i,1)~=0 &&
Bad_plot_Z(i,1)~=0
        Testbad_indi(i,1)=1;
    else
        Testbad_indi(i,1)=0;
    end

end
find_Testgood_indi=find(Testgood_indi);
siz_fin_Tesgod=size(find_Testgood_indi);
find_Testbad_indi=find(Testbad_indi);
siz_fin_Testbad=size(find_Testbad_indi);

siz_Previous_Genran = size(Previous_Genran);
Ran_siz_Previous=randperm(siz_Previous_Genran(1,1));

Persn_siz_Previous=siz_Previous_Genran(1,1)*0.05;
for i = 1:siz_Previous_Genran(1,1)
    for j = 1:siz_Previous_Genran(1,2)
        if siz_fin_Tesgod(1,1)>=Persn_siz_Previous

            for i1=1:Persn_siz_Previous
                Previous_Genran{find_Testbad_indi(i1),j}=
Previous_Genran{find_Testgood_indi(i1),j};
            end
        else
            for i2=1:siz_fin_Tesgod(1,1)
                Previous_Genran{find_Testbad_indi(i2),j}=
Previous_Genran{find_Testgood_indi(i2),j};
            end

        end
    end
end

for i = 1:siz_Previous_Genran(1,1)
    for j = 1:siz_Previous_Genran(1,2)
        NewGeneration{i,j}=Previous_Genran{Ran_siz_Previous(i),j};

    end
end
NewGeneration;
toc; %Measure performance using stopwatch timer
```

9 MATLAB CODES FOR IMPLEMENT THE CROSSOVER AND MUTATION OPERATORS IN VECTOR EVELUATED – GA (VEGA)

```
%%*****************************************************************
*****
%% Implementation: Vector - Evaluated GA
 %  Lalith P. Karunarathne, BEng(Hons)
 %  University Of Warwick, Coventry
%%*****************************************************************
*****
%% Vector - Evaluated GA (Crossover & Mutation operation)

tic %Measure performance using stopwatch timer
siz_Popula_VecEv=size(Population);
NubOfGen_VecEv=15;
Crosoverd_Popu_VecEv={};
Previous_Genran_VecEv={};
NewGeneration_VecEv={};
CrossoverNewGen_VecEv={};

for igv =1:NubOfGen_VecEv

    if igv==1
    Previous_Genran_VecEv=Population;  % same population

run('C:\Users\LALITHK\Documents\MATLAB\First_ReseEvolve\FitnessEvolu
Vector') %Vector Evaluated GA

    else
    %NewGeneration;

    for ip =1:siz_Popula_VecEv(1,1)
        for jp = 1:siz_Popula_VecEv(1,2)
            if ip <= (siz_Popula_VecEv(1,1)/2)
           Crosoverd_Popu_VecEv{2*ip-
1,jp}=NewGeneration_VecEv{(2*ip),jp};
           Crosoverd_Popu_VecEv{2*ip-1,5}=
NewGeneration_VecEv{(2*ip-1),5};
           Crosoverd_Popu_VecEv{2*ip-1,6}=
NewGeneration_VecEv{(2*ip-1),6};

        Crosoverd_Popu_VecEv {2*ip,jp}= Crosoverd_Popu_VecEv {(2*ip-
1),jp};
        Crosoverd_Popu_VecEv {2*ip,5}= Crosoverd_Popu_VecEv
{(2*ip),5};
        Crosoverd_Popu_VecEv {2*ip,6}= Crosoverd_Popu_VecEv
{(2*ip),6};
            end
        end
    end

%Mutation>>>>>
      for ip =1: siz_Popula_VecEv(1,1)*2
            for jp = 1: siz_Popula_VecEv(1,2)
```

```
          Muta_Crosoverd_Popu_VecEv {ip,jp}= Crosoverd_Popu_VecEv
{ip,jp};

            ra_Pernum_VecEv = randperm(siz_Popula_VecEv (1,2));
            ra_Pernum_VecEvx = randperm(siz_Popula_VecEv (1,1));
            ra_ Pernum_VecEvy = randperm(length(All_NeWok_Sources));

      Muta_Crosoverd_Popu_VecEv{ip, ra_Pernum_VecEv }(ra_
      Pernum_VecEvy,:)= ... Crosoverd_Popu_VecEv {ra_Pernum_VecEv x,
      ra_Pernum_VecEv}...     ( ra_ Pernum_VecEvy,:);

            end
      end

    if igv >1
        Previous_Genran_VecEv= Muta_Crosoverd_Popu_VecEv;

run('C:\Users\LALITHK\Documents\MATLAB\First_ReseEvolve\FitnessEvolu
Vector)

    end
    end

end
```

10 MATLAB CODES FOR IMPLEMENT THE SELECTOR OPERATOR, FITNESS ASSIGNMENT AND INDIVIDUAL EVALUATION FOR VECTOR EVALUATED - GA (VEGA)

```
%*******************************************************************
*****
%% Vector-Evaluated GA
%Fitness Evolution for Generation
%  Lalith P. Karunarathne, BEng(Hons)
 %  University Of Warwick, Coventry
%%******************************************************************
*****
%Previous_Genran

Num_Source=length(All_NeWok_Sources);
Trans_Population_VecEv={};
Trans_Pop_link_VecEv={};
Dub_Num_Source=2*Num_Source;

for i=1:Size_fi_Tet__Lin_Dis(1,2) %<-Number of sinks
    for j=1:Population_Size
        for i1=1:NumNodes
            for j1=1:Num_Source

Trans_Population_VecEv{j,i}(i1,j1)=Previous_Genran_VecEv{j,i}(j1,i1)
;
            end
        end
    end
end
```

```
odd_Num=(1:2:Dub_Num_Source);
even_Num=(2:2:Dub_Num_Source);
Row_Index=(2:NumNodes);
Fina_Trans_Pop_link_fist={};

for i=1:Size_fi_Tet__Lin_Dis(1,2)
    for j=1:Population_Size
        for i1=1:length(Row_Index)
            for j1=1:length(odd_Num)
                  Trans_Pop_link_VecEv{j,i}(i1,odd_Num(j1))=
Trans_Population_VecEv{j,i}(i1,j1);
                  Trans_Pop_link_VecEv{j,i}(i1,even_Num(j1))=
Trans_Population_VecEv{j,i}(Row_Index(i1),j1);
                  if Trans_Pop_link_VecEv{j,i}(i1,odd_Num(j1))~=0 &&
Trans_Pop_link_VecEv{j,i}(i1,even_Num(j1))~=0

Fina_Trans_Pop_link_VecEv{j,i}(i1,odd_Num(j1))=
Trans_Pop_link_VecEv{j,i}(i1,odd_Num(j1));

Fina_Trans_Pop_link_VecEv{j,i}(i1,even_Num(j1))=
Trans_Pop_link_VecEv{j,i}(i1,even_Num(j1));
                  end
            end
        end
    end
end

%Previous_Genran{1,1}
%NewGeneration=Previous_Genran;
%%  Fina_Trans_Pop_link_fist{j,i} convert to spares matrix
Vercat_Fina_Trans_Pop_link_VecEv={};
One_zero_Tran_Pop_VecEv={};
size_Fina_Trans_Pop_link_VecEv = size(Fina_Trans_Pop_link_VecEv);
size_Vercat_Fina_Trans_VecEv={};
Numof_links_individ_VecEv=[];
Weight_Indivi_VecEv={};
Weightof_links_individ_VecEv=[];

for i=1:size_Fina_Trans_Pop_link_VecEv(1,1)
      Vercat_Fina_Trans_Pop_link_VecEv{i,1}=
vertcat(Fina_Trans_Pop_link_VecEv{i,:});

size_Vercat_Fina_Trans_VecEv{i,1}=size(Vercat_Fina_Trans_Pop_link_Ve
cEv{i,1});
      for i1 =1:NumNodes
          for j1 =1:NumNodes
              One_zero_Tran_Pop_VecEv{i,1}(i1,j1)=0;
          end
      end
      for i2 =1:size_Vercat_Fina_Trans_VecEv{i,1}(1,1)
         for j2 =1:length(odd_Num)
             if
Vercat_Fina_Trans_Pop_link_VecEv{i,1}(i2,odd_Num(j2))~=0 &&
Vercat_Fina_Trans_Pop_link_VecEv{i,1}(i2,even_Num(j2))~=0

One_zero_Tran_Pop_VecEv{i,1}(Vercat_Fina_Trans_Pop_link_VecEv{i,1}(i
2,odd_Num(j2)),...

Vercat_Fina_Trans_Pop_link_VecEv{i,1}(i2,even_Num(j2)))=1;
```

```
             end
          end
      end

Numof_links_individ_VecEv(i,1)=sum(sum(One_zero_Tran_Pop_VecEv{i,1})
);

    for i1 =1:NumNodes
        for j1 =1:NumNodes
            if One_zero_Tran_Pop_VecEv{i,1}(i1,j1)==1 &&
Weight_Mtx(i1,j1)~=0
                Weight_Indivi_VecEv{i,1}(i1,j1)=Weight_Mtx(i1,j1);
            end
        end
    end

Weightof_links_individ_VecEv(i,1)=sum(sum(Weight_Indivi_VecEv{i,1}))
;
end

min_Numof_links_indi_VecEv=min(Numof_links_individ_VecEv);
min_Weightof_links_ind_VecEv=min(Weightof_links_individ_VecEv);

One_Zero_Coding_Colm_VecEv={};
for i=1:size_Fina_Trans_Pop_link_VecEv(1,1)
   for i1 =1:NumNodes
      for j1 =1:NumNodes
            One_Zero_Coding_Colm_VecEv{i,1}(i1,j1)=0;
      end
   end
for j2=1:length(All_Coding_Node)
     One_Zero_Coding_Colm_VecEv{i,1}(:,All_Coding_Node(j2))=
One_zero_Tran_Pop_VecEv{i,1}(:,All_Coding_Node(j2));
end
end
%One_Zero_Coding_Colm_fist{1,1}
%%
size_Rowx_VecEv ={};
size_Rowy_VecEv ={};
find_size_Rowx_VecEv={};
Num_Inlink_Coin_pt_VecEv={};
Codin_Pt_VecEv={};
Num_Inlink_each_Indi_VecEv={};
Test_One_zero_size_Rowx_VecEv={};
Average_inlinks_per_CodingPt_VecEv=[];
plot_X_VecEv=[];
plot_Y_VecEv=[];

%Pareto Fronts
% Object 1
ParetoFront_CodinPts_VecEv=2;
% Object 2
ParetoFront_Numof_links_indi_VecEv=min_Numof_links_indi_VecEv;
%Object 3
ParetoFront_Weightof_links_ind_VecEv=min_Weightof_links_ind_VecEv;
%ParetoFront_OneCP_perSink;

for i=1:size_Fina_Trans_Pop_link_VecEv(1,1)
```

```
    for j=1:NumNodes

[Rowx_VecEv,Colmx_VecEv]=find(One_Zero_Coding_Colm_VecEv{i,1}(:,j));
       size_Rowx_VecEv{i,1}(j,[1 2])=size(Rowx_VecEv);
       if size_Rowx_VecEv{i,1}(j,1)<2
          size_Rowx_VecEv{i,1}(j,1)=0;
       end
       %All inlinks at all coding points in each individual

%Num_Inlink_each_Indi_VecEv{i,1}=sum(size_Rowx_VecEv{i,1}(:,1));
       % coding points index
       Codin_Pt_VecEv{i,1}=find(size_Rowx_VecEv{i,1}(:,1));


Test_One_zero_size_Rowx_VecEv{i,1}(1,j)=size_Rowx_VecEv{i,1}(j,1);
       if Test_One_zero_size_Rowx_VecEv{i,1}(1,j)~=0
           Test_One_zero_size_Rowx_VecEv{i,1}(1,j)=1;
       end

Num_Inlink_Coin_pt_VecEv{i,1}(1,1)=sum(size_Rowx_VecEv{i,1}(:,1));
       [Rowy_VecEv,Colmy_VecEv]=find(size_Rowx_VecEv{i,1}(:,1));
       size_Rowy_VecEv{i,1}(1,[1 2])=size(Rowy_VecEv);
    end
    % Average inlinks per coding points for each individual

%Average_inlinks_per_CodingPt_VecEv(i,1)=Num_Inlink_each_Indi_VecEv{
i,1}(1,1)/size_Rowy_VecEv{i,1}(1,1);

    plot_X_VecEv(i,1)=size_Rowy_VecEv{i,1}(1,1); % Objective -1,
Number of coding point

    plot_Y_VecEv(i,1)=Numof_links_individ_VecEv(i,1);

    % good individual - objective 1
    if (size_Rowy_VecEv{i,1}(1,1)-ParetoFront_CodinPts_VecEv)>=0 &&
(size_Rowy_VecEv{i,1}(1,1)-ParetoFront_CodinPts_VecEv)<=2
        plot_X_VecEv(i,1)=size_Rowy_VecEv{i,1}(1,1);
    else
        plot_X_VecEv(i,1)=0;
    end



    % Good Individual - objective 2
    if (Numof_links_individ_VecEv(i,1)-
ParetoFront_Numof_links_indi_VecEv)>=0 &&...
            (Numof_links_individ_VecEv(i,1)-
ParetoFront_Numof_links_indi_VecEv)<=2
        plot_Y_VecEv(i,1)=Numof_links_individ_VecEv(i,1);
    else
        plot_Y_VecEv(i,1)=0;
    end



    % Good Individual - objective 3
    if (Weightof_links_individ_VecEv(i,1)-
ParetoFront_Weightof_links_ind_VecEv)>=0 &&...
            (Weightof_links_individ_VecEv(i,1)-
ParetoFront_Weightof_links_ind_VecEv)<=2
```

```
        plot_Z_VecEv(i,1)=Weightof_links_individ_VecEv(i,1);
    else
        plot_Z_VecEv(i,1)=0;
    end

end

%% Feasibility test "All sinks have been connected through at least
one
% coding node"
% convert "VerCat_Gen1_Gen2" into One_Zero_Test cell
%size_Rowy{50,1}
size_Codin_Pt_VecEv=size(Codin_Pt_VecEv);
Test_size_Codin_Pt_VecEv={};
Test_One_zero_Codin_Pt_VecEv={};
for i=1:size_Codin_Pt_VecEv(1,1)
    for j=1:NumNodes
        Test_One_zero_Codin_Pt_VecEv{i,1}(1,j)=0;
    end
   Test_size_Codin_Pt_VecEv{i,1}(1,[1 2])=size(Codin_Pt_VecEv{i,1});
   for i1=1:Test_size_Codin_Pt_VecEv{i,1}(1,1)

Test_One_zero_Codin_Pt_VecEv{i,1}(1,Codin_Pt_VecEv{i,1}(i1,1))=1;

   end
end
%Vercat_Fina_Trans_Pop_link{1,1}
%Test_One_zero_Codin_Pt{[1 2],1}

size_Previous_Genran_VecEv=size(Previous_Genran_VecEv);
size_Cell_Previous_Genran_VecEv=size(Previous_Genran_VecEv{1,1});
Test_MatrPrevious_Genran_VecEv=[];
%VerCat_Gen1_Gen2{1,2}
TestPrevious_Genran_VecEv={};
 TestPrevious_Cells_VecEv={};
 conca_TestPrevious_Cells_VecEv={};
 count_conca_Test_VecEv=[];
 count_Test_One_zero_Codin_VecEv=[];
Coding_Resou_Shar_ratio_VecEv=[];



for i=1:size_Previous_Genran_VecEv(1,1)
    for j=1:size_Previous_Genran_VecEv(1,2)
        for i1=1:size_Cell_Previous_Genran_VecEv(1,1)
            for j1=1:size_Cell_Previous_Genran_VecEv(1,2)
                TestPrevious_Cells_VecEv{i,j}(1,j1)=0;
              if Previous_Genran_VecEv{i,j}(i1,j1)~=0

TestPrevious_Genran_VecEv{i,j}(i1,Previous_Genran_VecEv{i,j}(i1,j1))
=1;
              end

TestPrevious_Genran_VecEv{i,j}(4,j1)=Test_One_zero_Codin_Pt_VecEv{i,
1}(1,j1);
              if TestPrevious_Genran_VecEv{i,j}(1,j1)==1 &&
TestPrevious_Genran_VecEv{i,j}(4,j1)==1||...
                 TestPrevious_Genran_VecEv{i,j}(2,j1)==1 &&
TestPrevious_Genran_VecEv{i,j}(4,j1)==1||...
```

```
                 TestPrevious_Genran_VecEv{i,j}(3,j1)==1 &&
TestPrevious_Genran_VecEv{i,j}(4,j1)==1

                 TestPrevious_Cells_VecEv{i,j}(1,j1)=1;

              end
              %TestPrevious_Cells{i,j}(1,j1) =
TestPrevious_Genran{i,j}(5,j1);

conca_TestPrevious_Cells_VecEv{i,1}(j,j1)=TestPrevious_Cells_VecEv{i
,j}(1,1);

count_conca_Test_VecEv(i,1)=nnz(conca_TestPrevious_Cells_VecEv{i,1})
;

count_Test_One_zero_Codin_VecEv(i,1)=nnz(Test_One_zero_Codin_Pt_VecE
v{i,1};

            end
        end
    end
end

for i=1:length(plot_Y_VecEv)
    if plot_X_VecEv(i,1)~=0
    subplot(3,2,1);plot(i,plot_X_VecEv(i),'ro');
    end

    if plot_Y_VecEv(i,1)~=0
    subplot(3,2,2);plot(i,plot_Y_VecEv(i,1),'g*');
    end

    if plot_Z_VecEv(i,1)~=0
    subplot(3,2,3);plot(i,plot_Z_VecEv(i,1),'b+');
    end

    if plot_X_VecEv(i,1)~=0  && plot_Y_VecEv(i,1)~=0 &&
plot_Z_VecEv(i,1)~=0

subplot(3,2,4);plot3(plot_X_VecEv(i,1),plot_Y_VecEv(i,1),plot_Z_VecE
v(i,1),'ko');
    end
    clear on
    grid on
    box on
    hold on

end

% Select subpopulation based on Objective 1
subpop_X=[];
subpop_Y=[];
subpop_Z=[];
Good_IndivXYZ=[];
for i=1:length(plot_Y_VecEv)
    if plot_X_VecEv(i,1)~=0
      subpop_X(i,1)=1;
    else
      subpop_X(i,1)=0;
    end
    if plot_Y_VecEv(i,1)~=0
      subpop_Y(i,1)=1;
    else
        subpop_Y(i,1)=0;
    end
    if plot_Z_VecEv(i,1)~=0
      subpop_Z(i,1)=1;
    else
        subpop_Z(i,1)=0;
    end
    if plot_X_VecEv(i,1)~=0  && plot_Y_VecEv(i,1)~=0
&&plot_Z_VecEv(i,1)~=0
        Good_IndivXYZ(i,1)=1;
    else
        Good_IndivXYZ(i,1)=0;
    end

end

fin_Good_IndivXYZ=find(Good_IndivXYZ);
siz_fin_Good_IndivXYZ=size(fin_Good_IndivXYZ);
if siz_fin_Good_IndivXYZ(1,1)>=4
    fprintf('Number of feasible individual
is',siz_fin_Good_IndivXYZ(1,1));
end

find_subpop_X=find(subpop_X);
siz_fin_subpop_X=size(find_subpop_X);

find_subpop_Y=find(subpop_Y);
siz_fin_subpop_Y=size(find_subpop_Y);

find_subpop_Z=find(subpop_Z);
siz_fin_subpop_Z=size(find_subpop_Z);
%Horizontal Concatination
Vertic_Cat=vertcat(find_subpop_X,find_subpop_Y,find_subpop_Z);
leng_Vertic_Cat=length(Vertic_Cat);
Trans_Vertic_Cat=[Vertic_Cat]';
Ran_leng_Vertic_Cat=randperm(leng_Vertic_Cat);

for i=1:leng_Vertic_Cat
    Rand_Vertic_Cat(i,1)=Vertic_Cat(Ran_leng_Vertic_Cat(i),1);
end

siz_Previous_Genran_VecEv = size(Previous_Genran_VecEv);
if leng_Vertic_Cat<siz_Previous_Genran_VecEv(1,1)
    dif_leng_Vertic=siz_Previous_Genran_VecEv(1,1)- leng_Vertic_Cat;
    for i1=1:dif_leng_Vertic
       Rand_Vertic_Cat(i1+leng_Vertic_Cat,1)=Rand_Vertic_Cat(i1,1);
    end
end

for i = 1:siz_Previous_Genran_VecEv(1,1)
    for j = 1:siz_Previous_Genran_VecEv(1,2)
```

```
NewGeneration_VecEv{i,j}=Previous_Genran_VecEv{Rand_Vertic_Cat(i,1),
j};

    end
end
NewGeneration_VecEv;
toc %Measure performance using stopwatch timer
```

APPENDIX C …

Evolutionary Minimization of Network Coding Resources

Lalith P. Karunarathne, Mark S. Leeson* and Evor L. HinesThe authors are with the School of Engineering, University of Warwick, Coventry, CV4 7AL, UK (e-mail: {l.p.karunarathne, Mark.Leeson, E.L.Hines}@warwick.ac.uk).

* Corresponding Author

Abstract — A method to identify feasible minimal network coding configurations between a source and a set of receivers without altering or modifying the established network infrastructure is proposed. The approach minimizes the resources used for multicast coding while achieving the desired throughput in the multicast scenario. Since the problem of identifying minimal configurations of a graph is known to be NP-hard, our method first identifies candidate minimal configurations and then searches for the optimal ones using a Genetic algorithm (GA). As the optimization process considers the number of coding nodes, the mean number of coding node input links and the sharing of resources by sinks, the problem is thus a multi-objective problem. Two multi-objective algorithms, MOGA and VEGA, are chosen to solve the problem because they are simple enough not to place heavy demands on source nodes when the minimal configuration is sought. The optimisation process is investigated by the simulation of a range of randomly generated networks of varying sizes. Performance differences between the multiple-objective GAs are observed which seem to arise from the difference in their methods of searching. Nevertheless, both methods perform well in terms of identifying feasible minimal configurations with optimised coding resources. The performance is assessed by comparing the optimised solutions with randomly chosen starting configurations. There are always reductions in the number of coding nodes used, typically of 50% and resource sharing is multiplied by several times. Typical mean in-link savings are 10% but may range from zero to close to 30%. We thus show that relatively simple multiple-objective GAs can deliver optimised minimal coding configurations for the network coding multicast problem. Moreover, the approach here offers an improvement over solutions in the literature since our method remains feasible for relatively large networks and its implementation at the source simplifies the functions that must be employed at intermediate nodes.

Index Terms— Coding resources, genetic algorithms, multicast, network coding, multi-objective optimization.

1. Introduction

Network multicast refers to the simultaneous transmission of the same information to multiple receivers in a network. Multicast transmission has historically been a demanding task that consumed considerable network resources such as channel bandwidth and network power. To minimize network resource usage, network coding (NC) [1] allows nodes to combine two or more independent bit streams via binary addition as well performing their traditional functions of packet routing and duplication. In the multicast NC problem, a source, *S*, needs to deliver *h* packets to *N* sinks over an underlying communication network ***G*.** Recently, considerable efforts have been made to minimize the coding resources in the multicast scenario [1], [2] and in this paper, a feasible engineering solution is proposed for this challenge. To illustrate the issues, the network shown in Figure 1 is considered with respect to a particular example scenario. Source *S* wishes to transmit a number of data packets, *say 3*, from s_1, s_2 and s_3, simultaneously to sinks t_1, t_2 t_3. *S* intends to identify a minimal configuration between itself and the sinks for its multicast traffic delivery.

In their seminal research, Ahlswede *et al.* [3] illustrated that if NC is permitted at the nodes of a network the communication rate can be improved over that obtainable by routing alone. Li *et al.* [4] showed that linear coding (in which each packet sent over the network is a linear combination of the original packets) is sufficient for multicast network coding problems. Koetter and Médard [5] introduced an algebraic framework for the study of network coding and gave a condition for valid codes. This framework was used by Ho *et al.* [6] to show that linear network codes can be efficiently constructed by employing a randomized algorithm. Jaggi *et al.* [7] proposed a deterministic polynomial-time algorithm to find feasible network codes for multicast networks.

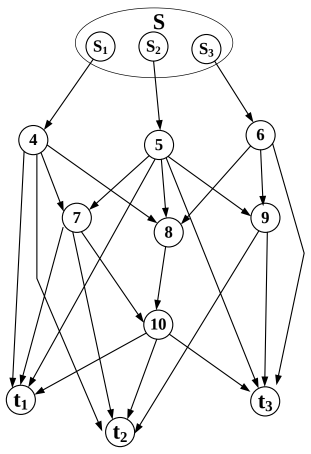


Figure 8: Example network used to illustrate the proposed method

The identification of the minimal configuration with a minimum number of coding points is NP-hard [1]. Here, our solution based on a Genetic Algorithm (GA) deals with this by developing candidate solution paths and identifying the best one rather that tackling the NP-hard problem. The remainder of the paper is organized as follows. Section 2 presents the problem formulation and Section 3 considers related work. The fourth and fifth sections introduce the proposed solution method and its simulation. Section 6 presents the results obtained followed by conclusions in Section 7.

2. Problem formulation

We consider a communications network represented by a directed acyclic graph $G = (V, E)$ with unit capacity edges and in which the value of the min-cut between the source node and each of the receivers is *h*. There is a set of *h* unit rate information sources $\{S_1, S_2 ......S_h\}$ and a set of *N* receivers $\{t_1, t_2 ......t_N\}$. We assume each receiver has at least one set of *h* linear disjoint paths (*h*-LDPs). We denote by (S_i, t_j), $1 \le i \le h, 1 \le j \le N$, a set of *h*-LDPs from the source to the receiver node *j* and the choice of paths is not necessarily unique. Our objective of interest is the minimal configuration $G' \in G$ with optimum coding resources, consisting of less than *hN* paths. We assume that source S_i, $1 \le i \le h$ simultaneously emits σ_i, $1 \le i \le h$ which is an element of some finite field $\mathbb{F}_\mathbf{q}$. In linear NC, each node of G' receives an element of $\mathbb{F}_\mathbf{q}$ from each input edge, and then forwards a linear combination of its inputs to each output edge.

Detailed discussion concerning the linear NC resources required for multicasting is contained in [1]. Therein, the major complexity components are described as *Set-up complexity* and *Operational complexity*. The former denotes the complexity of designing the network coding scheme, which includes selecting the paths through the information flows and determining the operations that the nodes of the network perform. The latter encompasses the running cost of using NC, that is the amount of computational and network resources required per information unit successfully delivered. Moreover, this complexity is strongly correlated with the NC scheme employed. To recover the source elements σ_i which have been linearly combined over $\mathbb{F}_\mathbf{q}$ by the coding nodes, each receiver needs to solve a system of h^2 linear equations, requiring $O(h^3)$ operations over $\mathbb{F}_\mathbf{q}$ if Gaussian elimination is used. The linear combination of *h* information streams requires $O(h^2)$ finite field operations. The complexity is further affected by the *size of the finite field* over which we operate. The cost of finite field arithmetic grows with the field size. For example, typical algorithms for multiplication or inversions over a field of size $q = 2^n$ require $O(n^2)$ binary operations [1]. Also the field size affects the required storage capabilities at intermediate network nodes. Moreover the complexity is affected by the *number of network coding points*, which are generally more expensive due to the need to equip them with encoding capabilities. In addition, coding points incur delay and increase the overall complexity of the network [8]. The computational complexity at each coding point of G' is considerably increased by the number of *in-links* per coding point which exhausts the coding resources via increasing the operational complexity of the network [1]. Therefore we are interested in optimizing the *number of coding points* and *the number of in-links per coding point* while identifying the minimal configuration G'.

3. Related work

This problem is somewhat similar to that of the "Travelling Salesman" [9] and in both cases GAs may be employed to search for the suitable geometrical properties (e.g. shortest paths, minimal configuration). Determining a minimal set of nodes where coding is required is known to be difficult [2]. The problem of deciding whether a given multicast rate is achievable without coding, i.e., whether the minimum number of required coding nodes is zero or not, reduces to a multiple Steiner subgraph problem, which is NP-hard [10]. Hence, the optimization problem to find the minimal number of required coding nodes is NP-hard. Even approximating the minimal number of coding nodes within any multiplicative factor or within an additive factor of $|V|^{1-\xi}$ is NP-hard [11].

As a first attempt at an evolutionary approach to the NC problem, Kim *et al.* [2] considered coding resource minimization while achieving the desired throughput in a multicast scenario by inspection of the outgoing links of all of the nodes. In this NP-hard problem they employed the structure of the standard GA, which was introduced by Holland [12], operating on a set of candidate solutions which it improved sequentially via mechanisms inspired by biological evolution (recombination and mutation of genes plus survival of the fittest). The algorithm proposed in [2] reduces the number of coding links/nodes relative to prior approaches in [1] and [8] and applies to a variety of generalized scenarios.

Here, our solution overcomes two major drawbacks of the approach in [2]. Firstly, a node where coding is required cannot be decided independently, which implies that whether coding is required at a node depends on whether coding is performed at other nodes; the verification procedure cannot thus be applied separately to each node. Hence, when the number of involved nodes is augmented, the complexity grows rapidly. Secondly, the GA operations are must run in each node on an individual basis meaning that costly functional integration (hardware and software upgrade) is essential at each node. These operations increase transmission complexity and exhaust network node resources.

4. Proposed Solution with Evolutionary Approach

Here, the proposed solution runs at the source where the processing and memory capacity are sufficient to execute these algorithms. All intermediate nodes are only required to perform their core operations (forwarding, duplicating and coding). Figure 2 shows a block diagram of the optimization process which comprises two fundamental elements: the preliminary process (which creates a search space) and the multi-objective GA process (which creates and identifies feasible minimal configurations), which will now be considered in turn.

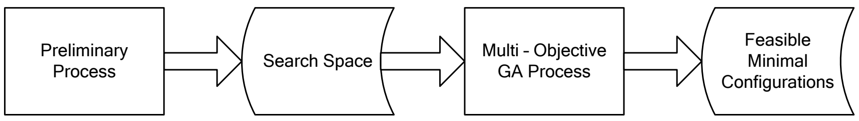


Figure 9: Block diagram of the operation of the proposed solution

4.1. Preliminary process

The preliminary process provides unevaluated individuals to the search space and then the two generic algorithms (path augmention and linear disjoint path) contribute to create the search space.

4.1.1. Path augmentation algorithm

This implementation is new but derives from the Breadth First Search (BFS) algorithm [13]. The algorithm identifies all available paths from each sub source $(\{S_i\}, 1 \le i \le h)$ to each receiver $(\{t_j\}, 1 \le j \le N)$. Figure 3(a) shows all available paths identified for receiver $\{t_1\}$. The algorithm's time complexity can be calculated as $O\left(h|V|^2\right)$, where $|V|$ is number of nodes in *G(V,E).*

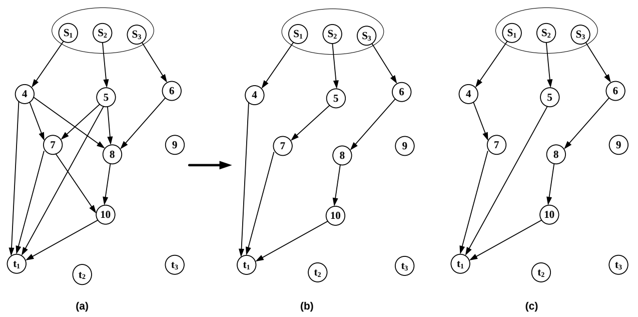


Figure 10: (a) All available paths from each sub source {*S₁, S₂, S₃*} to receiver {*t₁*}; (b) and (c) Two different sets of linear disjoint paths for receiver {*t₁*}.

4.1.2. Linear disjoint path algorithm

The set of *h* paths between the source and receiver is defined as a set of linear disjoint paths when none of the paths overlap. The algorithm hierarchically examines all available paths from each sub source $(\{S_i\}, 1 \le i \le h)$ to each receiver $(\{t_j\}, 1 \le j \le N)$ to form different sets of *h*-linear disjoint paths (LDPs). Figures 3(b) and 3(c) show two different sets of LDPs identified for receiver $\{t_1\}$. The two paths are compared to form a set of LDPs with time complexity $O(2|V|)$. Here the set of LDPs is classified based on sink IDs with the denomination $Gn_{t_j}^q$ where $(1 \le j \le N)$ for the q^{th} path to the sink. For example, sink $\{t_1\}$ has three sets of 3-linear disjoint paths: $Gn_{t_1}^1$, $Gn_{t_1}^2$ and $Gn_{t_1}^2$.

This algorithm contributes to the satisfaction of the multicast demand (min-cut max-flow) theorem and the formation of the minimal configuration. It is an enhancement to the approach in [1] for identifying the minimal configuration which considered edge disjoint paths only. Here, the LDP algorithm allows us to overcome this restriction so that overlapped paths are also accepted to form the minimal configuration. For example, in the minimal configuration of Figure 4 there are two different data streams $\{S_3\}$and $\{S_2\}$at link (8,10).

4.2. Search space

A random shuffle process picks $Gn_{t_j}^q$ from each sink column $t_j, (1 \le j \le N)$ and creates a row. A row is defined as an *individual* and its elements $Gn_{t_j}^q$ are defined as *genes*. The random shuffle process is terminated when the search space size (Z) reaches a pre-defined number. This is a significant stage of the proposed solution because it is the commencement of the mapping of NC problem into a GA framework.

$$\left[\!\left[\underbrace{\begin{Bmatrix} \langle S_1, 4, t_1 \rangle \\ \langle S_2, 5, 7, t_1 \rangle \\ \langle S_3, 6, 8, 10, t_1 \rangle \end{Bmatrix}}_{Gene1} \underbrace{\begin{Bmatrix} \langle S_1, 4, 7, t_2 \rangle \\ \langle S_2, 5, 8, 10, t_2 \rangle \\ \langle S_3, 6, 9, t_2 \rangle \end{Bmatrix}}_{Gene2} \underbrace{\begin{Bmatrix} \langle S_1, 4, 7, 10, t_3 \rangle \\ \langle S_2, 5, t_3 \rangle \\ \langle S_3, 6, 9, t_3 \rangle \end{Bmatrix}}_{Gene3} \right]\!\right] \Leftarrow Individual$$

Figure 11: The random shuffle process creates an unevaluated individual to form the search space

The minimal configuration is created by the sets of LDPs. During this creation either the same or different data streams overlap at some links. A tail node of an overlapped link becomes a coding node and other intermediate nodes become forwarding nodes. The vital point is that data streams are either coded or not, all sinks are able to simultaneously receive multicast data via LDPs. Moreover the source can assign linearly independent coding vectors for coding nodes, therefore all sink are able to form full rank decoding matrixes. Thus the minimal configuration is either feasible or not, it is not necessary to evaluate for a full rank state in contrast to the approach in [2], which can unmanageable as a result of the evaluation.

4.3. Brief introduction to GAs

The operation of GAs is on a set of candidate solutions, referred to as a *population*. Each solution is typically represented by bit strings, trees, graphs or any data structure adjusted to the problem being solved and known as a *chromosome*. Each of these is assigned a *fitness value* that measures how well it solves the problem at hand, compared with other chromosomes in the population. Typically, a new population is generated from the current one using three genetic operators: *selection*, *crossover* and *mutation*. Chromosomes for the new population are selected randomly (with replacement) in such a way that fitter ones are selected with higher probability. For crossover, the surviving chromosomes are randomly paired, and an exchange of bit string subsets takes place in each pair to create two offspring. Chromosomes are then subject to mutation, which refers to random flips of the bits applied individually to each of the new chromosomes. The process of evaluation, selection, crossover and mutation forms one *generation* in the execution of a GA. The above process is iterated with the newly generated population successively replacing the current one. The GA terminates when a certain stopping criterion is reached, e.g., after a predefined number of generations. GAs have been applied to a large number of scientific and engineering problems, including many combinatorial optimization problems in networks [14]-[16].

4.4. Potential of GAs to solve this problem

There are several aspects of this problem suggesting that GA-based methods may be promising candidates. Such approaches have worked well if the space to be searched is large but not known to be perfectly smooth or unimodal. Moreover, they will operate even if the space is not well understood [17], which makes traditional optimization methods difficult to apply. Here, the search space of our problem is not smooth or unimodal (two objective constraints are unknown) with respect to the number of sets of linear disjoint paths because each sink has different combination sets of the linear disjoint paths. The search space in this work consists of a large number of feasible or infeasible individuals which are created by the different combination sets of linear disjoint paths. An NP-hard problem results in which the individuals are not well understood. It should also be noted that, while it is hard to characterize the structure of the search space, once provided with a solution we can verify its feasibility (count the number of coding nodes, an average number of in-links per coding point and average resources shared per coding node) in polynomial time. Thus, if the use of genetic operations can suitably limit the size of the space to be actually searched a solution may be obtained relatively efficiently using the established procedures of GAs [17].

4.5. Multi-objective GAs

Multi-objective formulations are realistic models for many complex engineering optimisation problems such as minimising cost, maximising performance, maximising reliability and so on. The multiple objectives are generally conflicting, preventing simultaneous optimisation of each one. For example, in Section 4.10, the first objective to minimise the number of coding points may result in a conflict with the third objective in that one or more sinks may lose their coding resource sharing. The GA is a popular meta-heuristic approach that is particularly well suited for this class of problem. Therefore traditional GAs are customised to accommodate multi-objective problems by using specialised fitness functions. Konak et al. [18] present a comprehensive overview of multiple-objective optimization methods using GAs. Here we are interested in schemes with relatively simple implementations because the search space and fitness evaluation method of the proposed solution are complex. Moreover, the proposed solution is implemented at the source node and complex search methods would place unfeasible demands on source nodes. Thus, based on [18] we select two multi-objective methods: Multi-objective GA (MOGA) and Vector-evaluated GA (VEGA) and investigate how these methods perform on the problem formulated above.

4.5.1. Multi-objective GA (MOGA)

The well established single-objective GA described in Section 4.3 above is modified to find a set of multiple non-dominated solutions in a single run. The potential of the GA to simultaneously search different regions of a solution space means that a MOGA is a promising candidate to find a diverse set of solutions for difficult problems such as those that are non-convex. The crossover operator of the GA may exploit structures of good solutions with respect to different objectives to create new non-dominated solutions in unexplored parts of a *Pareto front.* Therefore GAs have been the most popular heuristic approach to multi-objective design and optimization problems.

4.5.2. Vector-Evaluated Genetic Algorithm (VEGA)

In this method, the selection operator of GA is modified so that at each generation a number of sub-populations is generated by performing proportional selection according to each objective function in turn. Thus, for a population size K and number of objectives q, each sub-population's size is K/q. These sub-populations are then shuffled together to obtain a new population of size K; each new generation is created by the usual GA operations of crossover and mutation.

4.5.3. GA operations and individual evaluation

An initial population (P_1) is obtained by randomly picking the individuals in the search space at t = 1. The GA operations form a new generation (P_2) from P_1 (generally $P_t \rightarrow P_{t+1}$). The population size (K) is constantly maintained at the size of P_1 throughout the GA. P_1 is evaluated as described in Section 4.10 and the highest fitness individuals are recombined by crossover to form an offspring population (Q_t).

4.6. Crossover

The number of genes in each individual depends on the number of sinks (*N)* which requested multicast data from the source. In this work, *single point crossover* is employed with a crossover point $\beta = [N \times pr_c]$, where $[u]$ represents the nearest integer to *u*. A value of 0.7 for pr_c was found to give good results after experimentation. Figure 5 shows the crossover operation with $\beta = [3 \times 0.7] = 2$ with the illustrative assumption that the Pareto optimal solution (described in Section 4.10) is *f*(2,2,3). The parents are selected based on their fitness and both are close to the Pareto optimum in Figure 7(a). The recombination forms the offspring in Figure 7(b), which exhibit higher fitness values and are thus are added to the offspring population Q_t. In this implementation, each generation results in the recombination of sets of parents to form sets of offspring. By iteratively applying the crossover operator, genes of "good" individuals are expected to appear more frequently in the population, eventually leading to convergence to an overall "good" solution.

$$\textit{Single Point Crossover} \Downarrow$$

$$\left[\!\!\left[\begin{Bmatrix} \langle S_1,4,t_1\rangle \\ \langle S_2,5,7,t_1\rangle \\ \langle S_3,6,8,10,t_1\rangle \end{Bmatrix} \begin{Bmatrix} \langle S_1,4,7,10,t_2\rangle \\ \langle S_2,5,t_2\rangle \\ \langle S_3,6,9,t_2\rangle \end{Bmatrix} \begin{Bmatrix} \langle S_1,4,t_3\rangle \\ \langle S_2,5,8,10,t_3\rangle \\ \langle S_3,6,9,t_3\rangle \end{Bmatrix} \right]\!\!\right] \Rightarrow \left[\!\!\left[\begin{Bmatrix} \langle S_1,4,t_1\rangle \\ \langle S_2,5,7,t_1\rangle \\ \langle S_3,6,8,10,t_1\rangle \end{Bmatrix} \begin{Bmatrix} \langle S_1,4,7,10,t_2\rangle \\ \langle S_2,5,t_2\rangle \\ \langle S_3,6,9,t_2\rangle \end{Bmatrix} \begin{Bmatrix} \langle S_1,4,t_3\rangle \\ \langle S_2,5,7,t_3\rangle \\ \langle S_3,6,9,t_3\rangle \end{Bmatrix} \right]\!\!\right]$$

$$f(3,2,3) \quad \Downarrow \quad \Uparrow \qquad\qquad\qquad f(2,2,3)$$

$$\left[\!\!\left[\begin{Bmatrix} \langle S_1,4,t_1\rangle \\ \langle S_2,5,t_1\rangle \\ \langle S_3,6,8,10,t_1\rangle \end{Bmatrix} \begin{Bmatrix} \langle S_1,4,7,10,t_2\rangle \\ \langle S_2,5,t_2\rangle \\ \langle S_3,6,9,t_2\rangle \end{Bmatrix} \begin{Bmatrix} \langle S_1,4,t_3\rangle \\ \langle S_2,5,7,t_3\rangle \\ \langle S_3,6,9,t_3\rangle \end{Bmatrix} \right]\!\!\right] \Rightarrow \left[\!\!\left[\begin{Bmatrix} \langle S_1,4,t_1\rangle \\ \langle S_2,5,t_1\rangle \\ \langle S_3,6,8,10,t_1\rangle \end{Bmatrix} \begin{Bmatrix} \langle S_1,4,7,10,t_2\rangle \\ \langle S_2,5,t_2\rangle \\ \langle S_3,6,9,t_2\rangle \end{Bmatrix} \begin{Bmatrix} \langle S_1,4,t_3\rangle \\ \langle S_2,5,8,10,t_3\rangle \\ \langle S_3,6,9,t_3\rangle \end{Bmatrix} \right]\!\!\right]$$

$$f(2,2,3) \qquad\qquad\qquad f(2,2,3)$$

$$(a)\ \textit{Parents} \qquad\qquad\qquad (b)\ \textit{Offspring}$$

Figure 12: Crossover operation showing (a) the parents and (b) the offspring

4.7. Mutation

The mutation operator introduces random changes into the characteristics of chromosomes. It is generally applied at the gene level. In typical GA implementations, the mutation rate (probability of changing the properties of a gene) is very small and depends on the length of the chromosome. Therefore, the new chromosome produced by mutation should not be that different from the original one. Mutation plays a critical role in GA. The crossover operator leads to population convergence by making the chromosomes in the population comparable. Mutation reintroduces genetic diversity back into the population and assists the search escape from local optima. In this implementation, the length of chromosome depends on a number of sinks. If a gene is randomly substituted by mutation, the original chromosome diverges significantly because the mutation rate (*1/N*) is extremely high. This issue is eliminated by substituting a single path in a randomly selected gene and inserting a random path without perturbing a linear disjoint feature of the gene, producing factor of *h* a reduction in the mutation rate.

$$\begin{array}{c}
\textit{Randomly selected gene} \Downarrow \\
\left[\left[\left\{\begin{array}{l}\langle S_1,4,t_1\rangle\\ \langle S_2,5,7,t_1\rangle\\ \langle S_3,6,8,10,t_1\rangle\end{array}\right\}\left\{\begin{array}{l}\langle\ \rangle\\ \langle S_2,5,t_2\rangle\\ \langle S_3,6,9,t_2\rangle\end{array}\right\}\left\{\begin{array}{l}\langle S_1,4,t_3\rangle\\ \langle S_2,5,7,t_3\rangle\\ \langle S_3,6,9,t_3\rangle\end{array}\right\}\right]\right] \\
in \Uparrow \qquad \langle S_1,4,7,10,t_2\rangle \\
\langle S_1,4,8,10,t_2\rangle \qquad \Downarrow out
\end{array}$$

Figure 13: Mutation operation

Figure 6 shows the mutation operation on one of the offspring in Figure 5(b). Gene 2 is randomly selected and its path $\langle S_1, 4, 7, 10, t_2\rangle$ is randomly substituted by another path whilst keeping the linear disjoint feature of the gene.

4.8. Selection

A selector operator plays a vital role in this work because it may pull the search to a narrow area of search space. The selector operator is connected with the individual evaluation, (Section 4.10). The selector operator selects K of the offspring in the offspring population Q_t based on their fitness and they are copied into the generation P_{t+1}, where K is the population size.

The selector operator performs differently in MOGA and VEGA. The GA operators of crossover and mutation work on a mating pool to form the offspring population Q_t. The selector operator creates two different mating pools for MOGA and VEGA. The generation P_t are assigned their fitness using the objective functions and they are evaluated using the Pareto optimal F_{t+1}^{OP}. The selector operator in MOGA concerns closer individuals to the Pareto optimal F_{t+1}^{OP} and the MOGA mating pool is filled by them. For example, the individuals I_1 and I_2 and are in Figure 7 (b). But the selector operator in VEGA concerns closer individuals to each objective of the Pareto optimal F_{t+1}^{OP} and the VEGA mating pool is filled by them. For example, the individuals from I_1, I_2, I_4 and I_5 are closer to $f_{t+1}^{OP}(X)$, the individuals I_1, I_2, I_4 and I_5 are closer to $f_{t+1}^{OP}(Y)$ and individuals I_1, I_2 and I_6 are closer to $f_{t+1}^{OP}(Z)$, and the VEGA mating pool is filled by them. Moreover the offspring population Q_t are evaluated using the Pareto optimal F_{t+1}^{OP} and the selector operator performs on Q_t as same as the selector operator on MOGA.

4.9. Termination criterion

If the source is able to identify w feasible multicast structures (individuals) then the search is terminated and the current population returned or else the process repeats from crossover and t becomes t+1. Moreover if the termination condition is not met during g generations, the entire population is removed and the process randomly re-initiated.

4.10. Evaluation of individuals

The fitness evolution process proposed concentrates on optimising the network coding resources in the multicast scenario by identifying the minimal configurations between sources and sinks, and contributes to coding resource optimisation in them. Three objective functions are employed to assign fitness values to individuals in the initial population or the mating pool:

1. Optimise the number of coding nodes in individuals - $f_I(X_i)$;
2. Achieve a desired throughput rate (constraining a number of in-links at each coding point) - $f_I(Y_j)$;
3. Optimally share coding resources in individuals - $f_I(Z_k)$.

The first two objectives optimise the network coding resources; the first and third optimise network resources. An optimum number of coding nodes are in multicast routes of the minimal configuration when it consumes the optimum coding resources when selected by the source for its multicast transmission. The use of coding nodes in multicast transmission automatically implies that a number of channels simultaneously convey more than one packet, contributing to efficient channel capacity use and network resource savings. The second objective allows the source to maintain a desired throughput rate during its multicast transmission. This can be achieved by constraining the number of input links at each coding point. Moreover it allows the saving of coding resources (storage capacity and computation) at the coding nodes. The third objective allows the sharing of the optimum coding resources with all sinks and may be enacted by considering the average coding resources sharing per coding node, defined as the sum of the number of receivers connected to each coding node divided by the number of coding nodes. In addition, it also improves the usage of the coding resources that are discovered via the first two objectives.

The problem is thus one of multi-objective optimisation and such cases generally exhibit conflicting objectives, preventing the simultaneous optimisation of each. In this case, the first and third objectives are in direct conflict since when the number of coding points is optimised, they are unlikely to be evenly spread. Here, the standard GA is customised to accommodate multi-objective problems by using specialised fitness functions and introducing methods to promote solution diversity. The approach is to determine an entire Pareto optimal solution set rather than a single fitness calculation in traditional GA. It is a most suitable solution because neither the first nor third objectives have pre-identified constraints. Therefore, the Pareto optimal solution is updated at each generation by comparing it with the one obtained in the previous generation.

It is assumed that the source intends to identify the *w* minimal configurations for termination. Minimal configuration *I* may be viewed as a point in the solution space $\{f_I(X_i), f_I(Y_j), f_I(Z_k)\}$. The points (X$_{OP}$, Y$_{OP}$, Z$_{OP}$) are objective constraints and the feasible set of them forms a Pareto optimal front. Figure 7(a) shows the objective constraints and Figure 7(b) shows a surface that is Pareto optimal on Z$_{OP}$. The Pareto optimal surface is updated at each generation with the first arising from the randomly selected initial population. The value of Y$_{OP}$ is maintained to be ≥2 but X$_{OP}$ and Z$_{OP}$ are updated at each generation with the minimum value being preferred for X$_{OP}$ and the maximum value for Z$_{OP}$. For example, Figure 7(b) shows Pareto optimal (*OP*$_{t-1}$, *OP*$_t$ and *OP*$_{t+1}$) for generation-$(t-1), t, (t+1)$ consecutively and they are updated at each generation. Pareto optimal set (*OP*$_{t-1}$) is: $\left(X_{PO}^{t-1}, Y_{PO}^{t-1}, Z_{PO}^{t-1}\right)$and set *OP*$_t$ is: $\left(X_{PO}^{t}, Y_{PO}^{t}, Z_{PO}^{t}\right)$. The comparison of (*OP*$_{t-1}$) and *OP*$_t$ is: $\left(\left[X_{PO}^{t-1} > X_{PO}^{t}\right], \left[Y_{PO}^{t-1} > Y_{PO}^{t}\right], \left[Z_{PO}^{t-1} = Z_{PO}^{t} = Z_1\right]\right)$ Therefore the Pareto optimal (*OP*$_t$) is moved to the position (*OP*$_{t+1}$) on surface Z$_1$.

The mutual comparison between individuals is extremely challenging in multi-objective optimisation and the proposed method can avoid the difficulty of comparison. At each selection operation, the individuals are assigned their fitness $\{f_I(X_i), f_I(Y_j), f_I(Z_k)\}$using objective functions. Then each individual is compared with the Pareto optimal (X$_{PO}$, Y$_{PO}$, Z$_{PO}$), using $f_C[(X_i - X_{OP}) \geq 0, (Y_j - Y_{OP}) \geq 0, (Z_k - Z_{OP}) \geq 0]$. If any individual is far away from Pareto optimal, it can be defined as a weakly fit or infeasible individual. With reference to Figure 7(b), the individual I$_3$ on surface Z$_1$ is in this position but individual I$_1$ is a fitter individual that should be selected in preference.

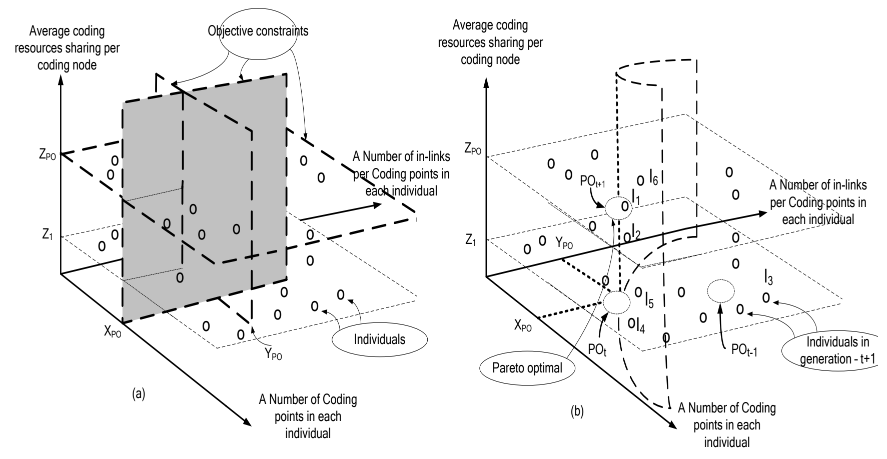

Average coding resources sharing per coding node
Objective constraints
Z$_{PO}$
Z$_1$
A Number of in-links per Coding points in each individual
X$_{PO}$
Y$_{PO}$
Individuals
(a)
A Number of Coding points in each individual
Average coding resources sharing per coding node
Z$_{PO}$
Z$_1$
PO$_{t+1}$
I$_1$
I$_2$
I$_3$
I$_4$
I$_5$
I$_6$
A Number of in-links per Coding points in each individual
Y$_{PO}$
X$_{PO}$
PO$_t$
PO$_{t-1}$
Pareto optimal
Individuals in generation - t+1
(b)
A Number of Coding points in each individual

Figure 14: Pareto optimisation process for the problem considered (a) objective constraints; (b) Pareto optimal front.

As shown in Figures 4 and 5, individuals are in a path-based format which is hard to analyse at the fitness assignment process stage. Therefore each individual is converted to a sparse matrix as shown in Figure 8, where {7, 8 and 10} can be identified as coding nodes because {7} is connected to {4} and {5}, which are in turn connected to sources {*S₁*} and {*S₂*}. Moreover {8} is connected to {*S₂*} and {*S₃*} via {5} and {6}. Node {10} is connected to coding nodes {7} and {8}. Then objective function $f_I(X_i)$ can thus be calculated as 3. The '1' entries of all coding nodes are counted via their respective rows and in Figure 8(b) there are six '1' entries in total, or an average of two per coding point (i.e. $f_I(Y_j) = 2$).

The objective function $f_I(Z_k)$ calculation process is to count when a sink column contains a '1' entry in a row representing a coding node. This means that the coding node contributes to the path to the sink in question. For example, in Figure 8(b) sink {t_1} has an entry '1' at row 7 meaning that the coding node 7 contributes to the route to {t_1}. However, when the first coding node identified is connected to a second coding node then this also adds to the number of nodes shared by the route. For example, in Figure 8(b) coding node 10 contributes to paths to {t_1, t_2, t_3} in its own right but because it is connected to coding nodes 7 and 8, these also contribute to routing to {t_1, t_2, t_3}. In total, the example has one contribution from {7} directly, three from {10} directly, and three each from {7} and {8} indirectly. Since there are three sinks, objective function $f_I(Z_k)$ can be calculated for Figure 8(b) as (1+3+3+3)/3 = 10/3 and the overall individual's fitness is *f*(3,2,10/3).

$$\left[\!\left[\begin{Bmatrix} \langle S_1,4,t_1\rangle \\ \langle S_2,5,7,t_1\rangle \\ \langle S_3,6,8,10,t_1\rangle \end{Bmatrix} \begin{Bmatrix} \langle S_1,4,7,10,t_2\rangle \\ \langle S_2,5,t_2\rangle \\ \langle S_3,6,9,t_2\rangle \end{Bmatrix} \begin{Bmatrix} \langle S_1,4,t_3\rangle \\ \langle S_2,5,8,10,t_3\rangle \\ \langle S_3,6,9,t_3\rangle \end{Bmatrix} \right]\!\right]$$

(*a*)

	S_1	S_2	S_3	4	5	6	7	8	9	10	t_1	t_2	t_3
S_1	0	0	0	1	0	0	0	0	0	0	0	0	0
S_2	0	0	0	0	1	0	0	0	0	0	0	0	0
S_3	0	0	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	1	0	1
5	0	0	0	0	0	0	1	1	0	0	0	1	0
6	0	0	0	0	0	0	0	1	1	0	0	0	0
7	0	0	0	0	0	0	0	0	0	1	1	0	0
8	0	0	0	0	0	0	0	0	0	1	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	1	1
10	0	0	0	0	0	0	0	0	0	0	1	1	1
t_1	0	0	0	0	0	0	0	0	0	0	0	0	0
t_2	0	0	0	0	0	0	0	0	0	0	0	0	0
t_3	0	0	0	0	0	0	0	0	0	0	0	0	0

(*b*)

Figure 15: (a) An individual in a path format; (b) A related sparse matrix

5. Simulation

The methods described were tested using a software environment developed in MATLAB R2009a and the computer system was: Windows Vista^{TM} Home Basic with service pack 2 (32 bit) running on an Acer Aspire 5735, Intel$^{®}$ Pentium$^{®}$ Dual CPU T3400 2.16GHz processor with 3GB RAM. Fifty different randomly generated topologies were used in the simulations. Each of these consisted of a single source with 3 data streams, and a different number of nodes, links and sinks. Table 1 provides the topological parameters which were used to create random networks. The tests proceeded as four projects, each of which consisted of a different number of runs. In each run, an equal size topology was employed but it was randomly generated upon commencement.

Table 3: Topological details and Parameter sets for Projects 1 – 4

Project	Runs	Topological Details			Parameter set $\{p_z, pr_c, pr_\mu, w\}$
		Nodes	Links	Sinks	
1	1-10	27	57	07	{100, 0.7, 0.05, 4}
2	11-20	30	68	07	{100, 0.7, 0.05, 4}
3	21-35	35	92	12	{100, 0.7, 0.03, 4}
4	35-50	40	113	17	{100, 0.7, 0.02, 4}

The parameters for the GA processes were: population size (p_z), crossover probability (pr_c), mutation probability (pr_μ) and termination criterion (*w*). They are represented as a parameter set $\{p_z, pr_c, pr_\mu, w\}$. The mutation probability was decreased with an increasing a number of sinks. Each simulation continued until either termination or a pre-defined generation number (*g* – here taken as 10 for all projects) had passed, in which case the GA is with a new initial population and marked as a failed search.

6. Results

These simulations do not attempt to deliver the actual multicast traffic levels rather identifying the minimal source to sink configurations, which is NP-hard. The performance of the proposed solution is considered in two parts, the preliminary process and the evolutionary process.

Figure 9 shows the performance of the two preliminary algorithms as a function of increasing scale (project) for the simulations in the all projects. The path augmentation is largely independent of network size in contrast to the linear disjoint path algorithm which has a more difficult task to perform as the network gets larger. Moreover, there also an inefficiency entering the discovery of disjoint paths because the algorithm obtains all available sets of linear disjoint paths but not all of these are needed in the discovery of the feasible minimal configuration.

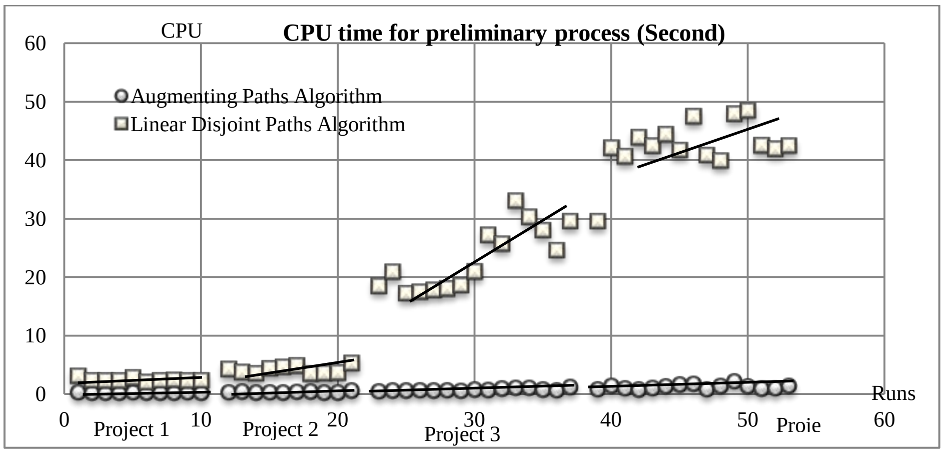


Figure 16: The performance of the two preliminary algorithms.

With respect to the evolutionary process, the search performance of the two multi-objective GA techniques MOGA and VEGA differed as the network size varied. Figure 10 shows the CPU time for MOGA and VEGA at each run of projects 1 to 4. In each run, an initial population was randomly selected and then used by both algorithms. An extremely high searching time (here 1000 seconds of CPU time) was taken as an indication of search failure as indicated in Figure 10. There is one subtle point to note in that the figure indicates that the two algorithms failed in runs 19, 22, 24, 32, 38, 39 and 41. However, on examination of their randomly generated topologies it was apparent that insufficient coding nodes had arisen from the stochastic nature of the generation process for the algorithm to ever find a solution. These runs were thus impossible from the outset and so were not taken into account when the simulation results were analysed.

Runs 1 to 10 were small scale networks and Figure 10 shows that VEGA was able to find feasible solutions for all runs in less than 100 seconds. MOGA succeeded in runs 1, 4, 5 and 7 with CPU times slightly below 200 seconds. The next set of ten runs considered slightly larger networks than the first ten. VEGA showed a slight degradation in its performance for these runs, with 7 out of 9 runs able to find a feasible solution in less than 200 seconds. MOGA showed a slight improvement in that it was able to return 5 out of 9 successful runs in less than 200 seconds. Runs 21-35 increased the network size further over runs 11-20 and VEGA's performance deteriorated further since it delivered only 4 out of 12 successful runs which took less than 200 seconds to complete. In contrast, MOGA showed a remarkable improvement, with 7 out of 12 runs succeeding in less than 200 seconds. Finally, runs 36-50 again used a network size that was larger than in runs 21-35. These tests took both algorithms closer to the limits of their operation with VEGA delivering just 2 successes out of 12 runs in less than 200 seconds and MOGA's corresponding success being reduced to 6 out of 12 runs.

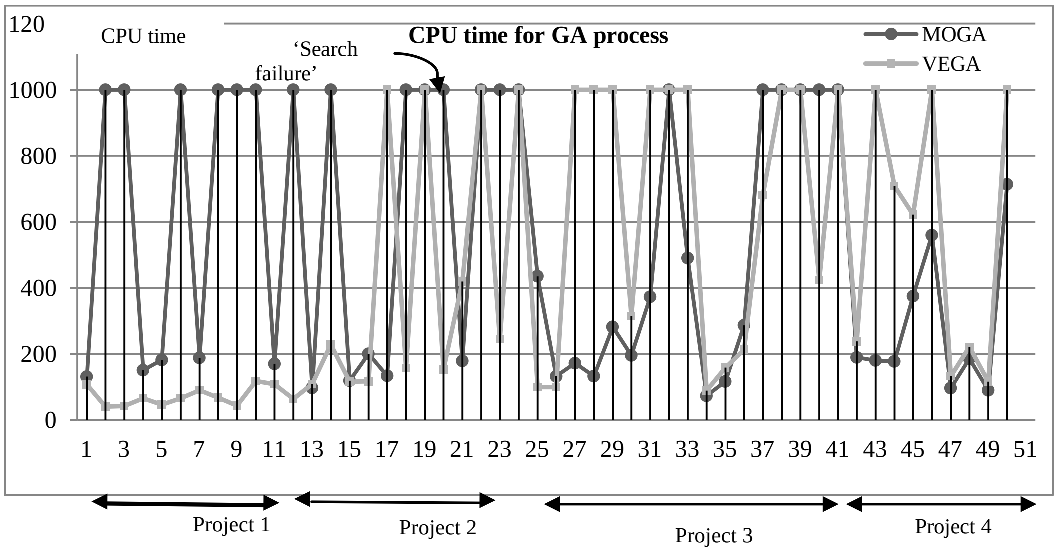


Figure 17: The performance evaluation of GA process base on CPU time

The searching potential of MOGA or VEGA depends not only on their performance but also on the availability of minimal configurations and coding nodes in the randomly generated networks (as was dramatically apparent in the apparent failures discussed above). The impact of this factor is shown in Figure 11, where the percentage success rates for MOGA and VEGA in the four projects are illustrated. In the first two, smaller, projects VEGA outperforms MOGA but this trend reverses as the network size grows in the second two projects. This is most likely explained by the fact that VEGA is a simple scheme and MOGA makes use of fitness sharing which promotes the exploration of new areas of the Pareto front by artificially reducing fitness of solutions in regions containing many solutions [18]. When the network is small, the number of solutions is reduced meaning that there is little to be contributed by the fitness sharing process. However, the increase in possible solutions with network size gives MOGA an advantage through its enhanced searching capabilities. The increasing number of nodes relative to sinks (and hence the increasing average node degree) coupled with the greater number of sinks as the projects progress greatly increases the routing possibilities permitting MOGA to gain an advantage from its greater sophistication.

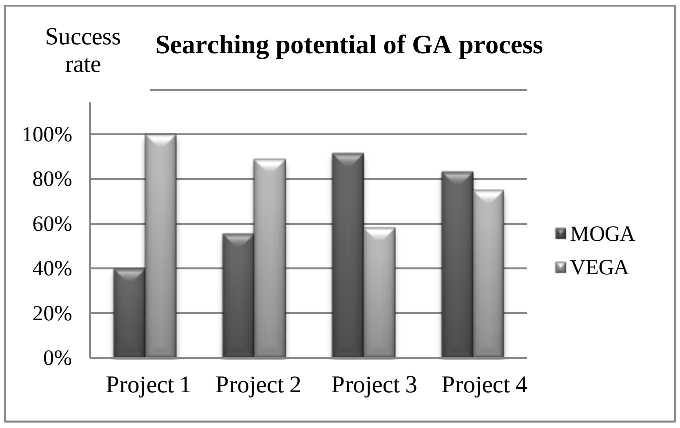


Figure 18: The performance evaluation of GA process based on searching potential

Table 2 shows fitness of most feasible individual (minimal configuration) identified at the end of randomly selected runs, fitness of most infeasible individual in an initial population of its run and their comparisons. Their comparisons prove how most feasible individual is advance than most infeasible individual in terms of network coding resources usage, if the source selects the most feasible minimal configuration for its multicast transmission. The savings in the number of coding nodes (X_i) and the mean number of in-links (Y_j) are given as percentages relative to the starting point. The resource sharing (Z_k) improvement is left as a ratio because all percentages would be extremely high and it is thus more useful to look at how the sharing has multiplied during the optimisation.

Table 4: A most feasible individual (minimal configuration) of each run compares with a most infeasible individual of its initial population based on their fitness. Randomly selected runs of each project are shown in the table.

Project	Randomly selected run	MOGA/ VEGA	Fitness of most feasible individual identified at the end of run F_{fI}			Fitness of most infeasible individual in initial population of run F_{iI}			Comparison F_{fI} and F_{iI}		
			$f_{fI}(X_i)$	$f_{fI}(Y_j)$	$f_{fI}(Z_k)$	$f_{iI}(X_i)$	$f_{iI}(Y_j)$	$f_{iI}(Z_k)$	Coding node saving	In-link saving	Resource sharing ratio
1	4	MOGA	2	2	6.5	5	2.4	1.4	60%	16.67%	4.64
		VEGA	2	2	6	5	2.4	1.4	60%	16.67%	4.28
	9	VEGA	2	2.5	7	6	2.67	0.83	66.67%	6.37%	8.43
	6	VEGA	3	2.33	5.33	5	2.8	1.2	40%	16.78%	4.44

	7	MOGA	2	2.5	6.5	6	2.83	1.66	66.67%	11.67%	3.91
		VEGA	2	2	6	6	2.83	1.66	66.67%	29.32%	3.61
2	13	MOGA	2	2	6	6	2.5	1.16	66.66%	20%	5.17
		VEGA	2	2.5	6.5	6	2.5	1.16	66.66%	0%	5.60
	17	MOGA	2	2.5	5.5	5	2.8	0.8	60%	10.71%	6.87
3	23	VEGA	3	2.33	7.66	7	2.85	1.14	57.14%	18.24%	6.72
	26	MOGA	3	2.67	6.67	6	2.83	1	50%	5.65%	6.67
		VEGA	4	2	7.25	6	2.83	1	33.33%	29.32%	7.25
4	42	MOGA	3	2.33	13.3	8	2.75	1.25	62.5%	15.27%	10.66
		VEGA	3	2.67	15.3	8	2.75	1.25	62.5%	2.91%	12.18
	40	VEGA	4	2.25	9.75	7	2.85	1.43	42.85%	21.05%	6.82
	46	MOGA	3	2	14.6	9	2.78	0.66	66.67%	28.06%	22.23

From Table 2, it may be observed that there are always savings in the number of coding nodes and these are generally 50% or more. Resource sharing is several times higher and may be improved by an order of magnitude. Mean in-link savings are more variable approaching nearly 30% at best but sometimes being at or near zero.

7. Conclusions

There are many situations where multicast is required and it has historically presented a demanding challenge in terms of network resources such as channel bandwidth and network power. The introduction of network coding offers the prospect of substantial reductions in resource requirements. The solution presented in this work comprises a preliminary process and a GA optimisation stage. The former deals with the aspects of path augmentation and linear disjoint path determination and produces a set of possible minimal configurations with optimised coding resources to deliver multicast traffic from the source to multiple sinks. These consist of three features (objectives) that contribute to optimise the network coding resources during multicast transmission. Searching for the optimum choices of minimal configurations is NP-hard so heuristic methods are needed. The search space and fitness evaluation processes are extremely complex in this problem placing restrictions on the complexity of the optimisation algorithms that may run within the network. Therefore, two of the classic multi-objective GAs, namely MOGA and VEGA, were chosen because their searching and implementation complexities are relatively low. The solution philosophy was to identify minimal configurations and chose the best from these thus sidestepping the difficult task of a full search of the complete solution space.

The performance of the algorithms proposed was investigated by simulating a range of networks of varying sizes. Specifically, for the preliminary processes path augmentation was undemanding in terms of CPU time and its operation was largely independent of network size. In contrast, the linear disjoint path algorithm placed greater demands on the CPU as the network size increased, reflecting that its task is more difficult as the network gets larger. Regarding the evolutionary optimisation, VEGA (the simpler of the two algorithms) exhibited better performance both in terms of CPU time and searching potential than MOGA for small networks but this position reversed as the network size grew. We believe this to be a result of the fitness sharing scheme present in MOGA that would not be of great utility for smaller search spaces. Nevertheless, both techniques exhibited good potential for identifying feasible minimal configurations with optimised coding resources. In most cases, there were considerable improvements in fitness by optimisation in comparison with randomly chosen starting configurations. All optimised cases delivered savings in the number of coding nodes, typically 50% and resource sharing was multiplied by several times in addition. Mean in-link savings of typically 10% usually resulted but the benefits ranged from zero to almost 30%.

Nevertheless, we have shown that relatively simple multiple-objective GAs can deliver optimised minimal coding configurations for the network coding multicast problem. Moreover, the approach here offers an improvement over solutions in the literature since our method remains feasible for relatively large networks and its implementation at the source simplifies the functions that must be employed at intermediate nodes. The approach taken has shown itself to be of great utility in minimizing complexity and resource demands, laying the foundations for efficient multicast network schemes for future traffic delivery.

8. Acknowledgement

This work was supported by EPSRC Grant EP/F033591/1, “Network Coding via Evolutionary Algorithms”.

9. References

[1] C. Fragouli and E. Soljanin, Network Coding Fundamentals, Foundations and Trends in Networking 2 (2007) 1-133.

[2] M. Kim, C. W. Ahn, M. Médard, and M. Effros, On minimizing network coding resources: An evolutionary approach, Network Coding Workshop (NetCod), 2006.

[3] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, Network information flow, IEEE Trans. Information Theory 46 (2000) 1204-1216.

[4] S.-Y. R. Li, N. Cai and R. W. Yeung, Linear Network Coding, IEEE Trans. Information Theory 49 (2003) 371-381.

[5] R. Koetter and M. Médard, An algebraic approach to network coding, IEEE/ACM Trans. Networking 11 (2003) 782–795.

[6] T. Ho, R. Koetter, M. Médard, D. Karger, and M. Effros, The benefits of coding over routing in a randomized setting, Proc. IEEE Int. Symp. Information Theory, Yokohama, Japan, 2003, 442.

[7] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L.Tolhuizen, Polynomial time algorithms for multicast network code construction, IEEE Trans. Information Theory 51 (2005) 1973-1982.

[8] M. Langberg, A. Sprintson and J. Bruck, The Encoding Complexity of Network Coding, IEEE Trans. Information Theory 52 (2006) 2386-2397.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, second ed., MIT Press, Cambridge Massachusetts, 2001.

[10] M. B. Richey and R. G. Parker, On multiple Steiner subgraph problems Networks 16 (1986) 423–438.

[11] M. Langberg, A. Sprintson, and J. Bruck, Network coding: A computational perspective, IEEE Trans. Information Theory 55 (2009) 147–157.

[12] H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

[13] K. Mehlhorn and P. Sanders, Algorithms and Data Structure, ACM Computing Classification, 173 (2008) 1-300.

[14] J. Arabas and S. Kozdrowski, Applying an Evolutionary Algorithm to Telecommunication Network Design, IEEE Trans. Evolutionary Computation 5 (2001) 309-323.

[15] G. Vidyarthi, A. Ngom and I Stojmenovic, A hybrid channel assignment approach using an efficient evolutionary strategy in wireless mobile networks, IEEE Trans. on Vehicular Technology 54 (2005) 1887-1895.

[16] Y. S. Kavian, H. F. Rashvand, W. Ren, M. S. Leeson, E. L. Hines and M. Naderi, RWA problem for designing DWDM networks – delay against capacity optimization, Electronics Letters 43 (2007) 892-893.

[17] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1996.

[18] A. Konak, D. W. Coit and A. E. Smith, Multi-objective optimization using genetic algorithms: A tutorial, Reliability Engineering and System Safety 91 (2006) 996-1007.