A Practical Network Coding and Routing Scheme based on Maximum Flow Combination

Lianlong Wu

BSc (Hons) Computer Science

*School of Computing and Intelligent Systems*

*University of Ulster, United Kingdom*

Supervisor: Kevin Curran

*April 2010*

# *Table of Contents*

[*Table of Contents* 2](#_Toc260081600)

[*List of Figures* 6](#_Toc260081601)

[*Acknowledgements* 7](#_Toc260081602)

[*Abstract* 8](#_Toc260081603)

[1 Introduction 9](#_Toc260081604)

[2 Network Coding 13](#_Toc260081605)

[2.1 Challenges in Network Coding 14](#_Toc260081606)

[2.2 Current Research Focus 15](#_Toc260081607)

[2.2.1 Centralized Network Coding 15](#_Toc260081608)

[2.2.2 Distributed Network Coding 15](#_Toc260081609)

[2.3 Applications 16](#_Toc260081610)

[3 Network Coding Fundamental Theory 19](#_Toc260081611)

[3.1 Definition of Graph 19](#_Toc260081612)

[3.2 Min-Cut and Max-Flow 19](#_Toc260081613)

[3.3 Min-Cut Max-Flow Theorem 21](#_Toc260081614)

[3.4 Multicast Problem 21](#_Toc260081615)

[3.5 Max-flow Bound Theorem 21](#_Toc260081616)

[3.6 Linear Network Coding 23](#_Toc260081617)

[3.7 Random Linear Coding 24](#_Toc260081618)

[3.8 Routing for Network Coding 24](#_Toc260081619)

[3.8.1 Coding at Merging Node 24](#_Toc260081620)

[3.8.2 Multicast Coding Routing 24](#_Toc260081621)

[4 Requirement Analysis and Specification 26](#_Toc260081622)

[4.1 Problem Description 26](#_Toc260081623)

[4.2 Algorithm Design Objectives 26](#_Toc260081624)

[4.3 Implementation Specification 26](#_Toc260081625)

[4.4 Network Example 27](#_Toc260081626)

[5 Algorithm Design for the Coding - Routing Scheme 29](#_Toc260081627)

[5.1 Construction of Multicast Flow Graph 29](#_Toc260081628)

[5.1.1 Max-Flow Algorithms 29](#_Toc260081629)

[5.1.2 Labelling Algorithm 29](#_Toc260081630)

[5.1.3 Merging Max-Flow for Obtaining Multicast Flow Graph 31](#_Toc260081631)

[5.1.4 The Target Set of an Edge 33](#_Toc260081632)

[5.2 Packet Representation and Operations 35](#_Toc260081633)

[5.2.1 Representation of Packet 35](#_Toc260081634)

[5.2.2 Split Target Set of Packet 36](#_Toc260081635)

[5.2.3 Coding Data Blocks of Two Packets 37](#_Toc260081636)

[5.3 Algorithm for Construction of Coding and Routing Scheme 37](#_Toc260081637)

[5.3.1 Node Arrangement in Topological Order 37](#_Toc260081638)

[5.3.2 Forward Operation 38](#_Toc260081639)

[5.3.3 Multicast Operation 39](#_Toc260081640)

[5.3.4 Coding Operation 41](#_Toc260081641)

[5.3.5 Coding and Routing Scheme Example 43](#_Toc260081642)

[5.3.6 Identification of Node Types 43](#_Toc260081643)

[6 Implementation of the Proposed Algorithm 45](#_Toc260081644)

[6.1 Development Environment 45](#_Toc260081645)

[6.1.1 Python Programming Language 45](#_Toc260081646)

[6.1.2 Graph Visualization with Graphviz 45](#_Toc260081647)

[6.2 Data Structures 45](#_Toc260081648)

[6.3 Data File Format 47](#_Toc260081649)

[6.3.1 Input File format 47](#_Toc260081650)

[6.3.2 Output File Format 47](#_Toc260081651)

[6.4 Flow Chart of Program 49](#_Toc260081652)

[6.5 Code Examples 51](#_Toc260081653)

[6.5.1 Codes for Drawing Figures 51](#_Toc260081654)

[6.5.2 Code for Topological Sorting 52](#_Toc260081655)

[7 Testing and Experimental Results 53](#_Toc260081656)

[7.1 Test Cases for Max-Flow Module 53](#_Toc260081657)

[7.2 Test Cases for Proposed Algorithm 55](#_Toc260081658)

[7.2.1 Typical Test Cases 55](#_Toc260081659)

[7.2.2 Random Test Cases 59](#_Toc260081660)

[7.3 Validation of the Coding and Routing Scheme 61](#_Toc260081661)

[7.3.1 Capacity Validation 61](#_Toc260081662)

[7.3.2 Decoding Validation 61](#_Toc260081663)

[7.4 Experimental Results 62](#_Toc260081664)

[7.4.1 Optimizing Number of Coding Nodes 62](#_Toc260081665)

[7.4.2 Comparison Number of Coding Links 65](#_Toc260081666)

[7.4.3 Comparison Running Time 66](#_Toc260081667)

[8 Conclusion 67](#_Toc260081668)

[9 References 68](#_Toc260081669)

# *List of Figures*

[*Figure 1.* Butterfly Network Coding Scheme 10](#_Toc260081670)

[*Figure 2.* Network Coding on Wireless Network Communication Application 11](#_Toc260081671)

[*Figure 3.* Packet Loss Example 13](#_Toc260081672)

[*Figure 4.* Time-Parameterized Graph of Avlanche type system 17](#_Toc260081673)

[*Figure 5.* Two sink nodes network with two max-flow examples 20](#_Toc260081674)

[*Figure 6.* Butterfly Network Routing Examples 25](#_Toc260081675)

[*Figure 7.* Unit capacity three sink node network coding scheme 28](#_Toc260081676)

[*Figure 8.* Max-flows to each sink node of butterfly network 31](#_Toc260081677)

[*Figure 9.* Merged max flows for butterfly network 33](#_Toc260081678)

[*Figure 10.* Target sets for butterfly network 34](#_Toc260081679)

[*Figure 11.* Target set for double capacity edge of butterfly network 35](#_Toc260081680)

[*Figure 12.* Forward operation at source node 39](#_Toc260081681)

[*Figure 13.* Multicast operation example 41](#_Toc260081682)

[*Figure 14.* Coding operation example 42](#_Toc260081683)

[*Figure 15.* Coding scheme with packet target set 43](#_Toc260081684)

[*Figure 16.* Colour scheme for different max-flows 48](#_Toc260081685)

[*Figure 17.* Node shapes of output images 48](#_Toc260081686)

[*Figure 18.* Arrow shapes of output images 48](#_Toc260081687)

[*Figure 19.* Work Flow Chart of Implementation 50](#_Toc260081688)

[*Figure 20.* Double capacity butterfly network test case 56](#_Toc260081689)

[*Figure 21.* Coding at source node test case 56](#_Toc260081690)

[*Figure 22.* Two coding nodes test case 57](#_Toc260081691)

[*Figure 23.* No coding operation test case 57](#_Toc260081692)

[*Figure 24.* Three sink nodes test case 58](#_Toc260081693)

[*Figure 25.* Redundancy capacity test case 58](#_Toc260081694)

[*Figure 26.* Random multicast network 60](#_Toc260081695)

[*Figure 27.* Minimal cost network coding example (Tao et al., 2008) 64](#_Toc260081696)

[*Figure 28.* Minimal cost coding scheme 64](#_Toc260081697)

# *Acknowledgements*

I would like to thank my supervisor Dr Kevin Curran for helping me with the research topic and for guiding me through every step of the research with great expertise and patience.

Una McGinley, a librarian in the University of Ulster, kindly ordered newly published books in my field on my behalf thus helping me with my research.

Finally, I am deeply grateful to my family for their strong support encouragement.

# *Abstract*

Network coding is a novel field of information theory and coding theory. It is a breakthrough over the traditional store-and-forward routing methods by allowing coding two or more packets together. From an information flow aspect, multiple flows could be overlapped in a routing scheme. Hence, the theory upper bound of multicast capacity could be achieved by network coding. In this project, a complete routing and coding scheme is constructed to realize the maximum multicast transportation task.

In order to obtain the scheme, the paths of multiple max-flows are worked out. Edges are divided into overlapped and normal type based on the merged max-flows. The transmitting data is represented using packets in a specific format. Multicast, forward and coding operations are defined to transmit data at the nodes. The nodes are classified according to the type of operations. A dynamic coding and routing algorithm is proposed to route packets gradually from source node to destinations in topological sorting order by the three operations on the path of merged max-flows. Based on simulation of random and real network benchmark topologies, it shows that network coding is necessary at a few nodes and links, the number of coding node and link is less than widely used rule and the result of evolutionary approach. Furthermore, it was proved that the use of simple *xor* operation could satisfy most of the network topologies. The running time of the algorithm presented here is less than one second for most of the benchmark and random data sets.

# Introduction

In the past half century, information flow being transferred in the network was following a way similar to the highway transportation or the water pipe systems in the real life. Packets in the digital network or signals in the analogue network were transmitted independently under different transportation stream without overlap. However, unlike the cars or water in the world, the information could be recombined in the transport systems so that the path of different information could be overlapped.

Network Coding (NC) is a recent research area in information theory (Ahlswede et al., 2000). It dramatically changes the traditional information processing and transportation style. For example, in the traditional computer networks, information packets are transmitted from the source through intermediate nodes by store-and-forward method. There is no extra processing except replication. With network coding, the operations such as *xor* or linear combinations between two or more different packets are allowed to join different information flows, and the original binary packet could be recombined or extracted later in the receivers. In the other words, data streams are not necessary to be kept independently in the communication networks.

The classical butterfly network model is shown in . One source node *S* intends to send both packets *a* and *b* to the target node *T1* and *T2*. For the traditional store and forward method demonstrated in Figure 1(*a*), two channels are required between *V3* and *V4*. By using the *xor* coding as shown in Figure 1(*b*), the bandwidth between *V3* and *V4*is reduced to one channel. Node *T1* receives packets *a* and *a*⊕*b*, and packet *b* could be obtained by *a*⊕(*a*⊕*b)*. Similar packet *a* could be decoded at *T2* with *b* and *a*⊕*b*. Hence, half of the bandwidth between *V3* and *V4* is saved.

*a*

*a*

*a*

*a*

*b*

*b*

*b*

*b*

*b*

*a*

S

T1

T2

V1

V2

V3

V4

*a*

*a*

*a*

*b*

*b*

*b*

T1

T2

V1

V2

V3

V4

(*a*)

(*b*)

*a*⊕*b*

*a*⊕*b*

*a*⊕*b*

S

1. Butterfly Network Coding Scheme

Another example of the wireless satellite multicast communication is shown in . Node *S* stands for the satellite, node *V1* and *V2* are two ground stations. *V1* sends packet *a* to *V2* through the satellite *S* and *V2* sends packet *b* inversely. By traditional methods, four time units are required:

1. *V1* sends *a* to *S*;
2. *V2* sends *b* to *S*;
3. *S* sends *a* to *V2*;
4. *S* sends *b* to V1.

If satellite broadcast *a*⊕*b* to both stations at the same time, each station could decode the other packet with the one it send. Thereby one time unit is saved.

S

V1

V2

*t*=1

S

V1

V2

*t*=2

S

V1

V2

*t*=3

*a*

*b*

*a*⊕*b*

*a*⊕*b*

1. Network Coding on Wireless Network Communication Application

With Network Coding, the theory max flow upper bound indicated by Shannon can be achieved. It was proved that applying Linear Network Coding might help to achieve the multicasting upper bound under one source multiple receivers’ situation . Linear Network Coding combines the coding and routing together from the physics layer and network layer respectively. Two problems need addressing in order to apply Network Coding in Multicasting:

1. Establish the routing for transportation
2. Adopt the coding pattern or code algorithm

Some efficient methods have been proposed to resolve the second issue. Random Network Coding (RNC) is a distributed implementation and widely used in current research. Compared with other coding approaches, such as heuristic, exponential or polynomial algorithms, Random Network Coding has lower complexity and could be easily applied in real network systems.

Little research has been done about the multicast routing issue and most efforts have concentrated to date on the theory aspect, such as demonstrating the feasibility of Network Coding theoretically or how to apply the coding on a specific network. However, the establishment of routing is a precondition of the network multicasting. In the traditional IP multicasting, the routing is set up through multicast Steiner Tree . The next sections are aimed at finding out feasible methods to determine the routing for network coding in multicast communications and construct the complete network coding scheme.

# Network Coding

The benefits of network coding are increasing multicast throughput, reducing resources usage and enhanced system robustness.

The butterfly network example in Figure 1 has shown the gains in network throughput. With network coding, the multicast task of two packets can be achieved in one time unit. Hence the transportation rate for each receiver is 2. It is as if each receiver has the network to themselves only. Without network coding, it requires 3 time units to finish the same multicast task of two packets as the contention between node *V3* and *V4* requires one more time unit to complete the transportation. So the transportation rate for each receiver is 1.5 It has been shown that the larger the degree of each node, the more improvement of the throughput by network coding (Wu et al., 2004).

The satellite application in Figure 2 is an example of optimizing wireless resources.

*rAB*=0.1

*rBC*=0.2

1. Packet Loss Example

Network coding also has the benefit of reducing packet loss as well. As shown in the above example, assume that packet loss rate *rAB* between node *A* and *B* is 0.1 and *rBC* equals to 0.2. With packet-level Forward Error Correcting (FEC) such as fountain codes, information could be transported from A to C at a success rate of (1-*rAB*)(1-*rBC*) = 0.72. If node B could decode and re-encode all the packets it received, then overall success rate could be increased to 1-rBC = 0.8, which is the same as maximum flow form node A to C. (Pakzad et al., 2005)

In addition, by applying network coding, the side effect caused by the failure of network links or nodes is minimized. So that the robustness of the network link is enhanced and the cost for network management is reduced (Ho et al., 2005).

## 

## Challenges in Network Coding

The challenges of network coding are in the following aspects.

For the security issue, by the nature of recombined packets and overlapped routes, the risk of wiretapping attacks is highly reduced. On the other hand, the operations of data in the intermediate nodes might affect the authenticity of the data.

In order to implement Network Coding, a high quantity of computation is required at every node in the network. For certain problems with limited conditions, the upper bound of the computation can be determined. However, for universal issues, this upper bound cannot be determined at this moment, and it is confirmed to be considerably large.

As the computational processing resource is getting cheaper under Moore’s law, the network bandwidth becomes the critical limitation. Network Coding could dramatically increase the network throughput by using the feasible computational power.

In a lot of networks, not all of them can be extracted as acyclic or directed graph, e.g. Token Ring networks. For the cyclic graph, the constructed coding is changing all the time. It is a challenge to construct coding scheme for such a graph.

The synchronization problem deserves attention as the encoder of the middle nodes might receive not only one input data flow in a short time period. For example, the node *V3* in Figure 1 requires synchronism before operation on two incoming packets. Synchronism is sensitive especially for real time application such as live voice or video transportation.

In an actual application of multicasting such as media stream publication, the nodes are usually changing all the time. As a result, the coding scheme is required to be rebuilt and the decoding for other nodes is effected as well. A scalable architecture is required for this dynamic changing situation.

## Current Research Focus

A lot of attention has been devoted the architecture for network coding in various types of networks and connections, in order to archive the maximum multicast capability. Generally speaking, the algorithms could be classified in centralized network coding and distributed network coding.

### Centralized Network Coding

There are two multicast centralized network coding algorithm. One is an algebraic structure by Koetter and Medard in 2002. It extends the previous conclusion to universal network and enhances the robustness. This new conclusion has been proved with ringed directed graph by max flow min cut theorem. The entire topology structure is required for constructing. One transformation matrix is used to represent the relation between input information from source and the information received at the node. The network coding is realized by constructing such a suitable transformation matrix.

The other multicast centralized network coding algorithm is a polynomial time complexity algorithm by Sanders in 2003. It simplified the construction for single source multicasting in no ring and no time delay graph. The required path set is pointed out by max flow min cut algorithm. Based on the set, the operation for each node is determined. This method not only reduces the construction algorithm from exponential to polynomial, but also reduces the lower bound of the alphabet required. In addition, Fragouli indicates that it is a colour painting problem for two source multicast network coding (Fragouli et al., 2004).

### Distributed Network Coding

Distributed network coding methods do not require the network topology information. The previous operation history is added to the packets so that the receiver could decode the original content from source. For example, the Random Network Coding adds the randomly generated coefficients to the head of each packet so that the sink node could decode packets without the knowledge of the entire network topology. It was shown that the success rate for decoding is acceptable, if the field size is large enough (Ho et al., 2003).

## Applications

Network Coding can be used in applications like Peer to Peer (P2P) Networks, Wireless Ad Hoc and others. Moreover, the related theory and application research contributes to Informatics, Coding Theory, Complexity Theory, Graph theory, Matrix Theory and other subjects.

Application Layer multicast is an alternative method to the Network Layer Multicast. While the information in network layer multicast is forwarded by the router, the information in application layer multicast is forward by the terminal host (PC or server). As the host usually has powerful computational resources, it is a compatible environment for network coding.

One typical application is P2P file sharing software Avalanche developed by Microsoft (Gkantsidis & Rodriguez, 2004). The file is divided into *n* blocks and encoded with Random Linear Coding at every host. As the linear coefficients are added to the encoded blocks, one host could decode the entire file once sufficient coded blocks are received. Meanwhile, the possibility of complete delivery is increased despite the joining or leaving of hosts dynamically. In fact, Avalanche applies network coding on a unique time-parameterized graph as shown in Figure 4, rather than the physical network (Raymond, 2008). The variable *t* denotes the unit time and the link stands for the delivery of packets between hosts.

Sender

*t*=0

*t*=1

*t*=2

*t*=3

*t*=4

Client Node

Client Node

Receiver

1. Time-Parameterized Graph of Avlanche type system

Due to the unreliability and multicasting feature in the physical layer of wireless network, network coding could resolve the issue in the traditional routing and cross-layer design. Applied network coding in the wireless network could increase the multicast throughput, reduce the number of hops and decrease the energy required for emission. In particular, with Random Network Coding, the original data could be retrieved at the terminal nodes even if some of the intermediate nodes or links are disabled.

One throughput optimization framework is proposed for multi-hop networks across network and physical layers. The network coding scheme provides for data routing and wireless power allocation (Yuan et al., 2005). Another experimental on 802.11 hardware shows that one XOR-only mechanism nearly doubles the network throughput (Katti et al., 2005).

Another reduced complexity network coding is proposed for ad hoc networks. The links are divided into two types: (*a*) entering relay nodes and (*b*) entering targets. The same capacity can be achieved by applying network coding only on the type (*a*) nodes and keeping the traditional routing style at all the type (*b*) nodes (Wu et al., 2005).

The minimum energy per bit can be achieved by network coding for mobile and ad hoc networks with linear program (Wu et al., 2004). Finally, a simple distributed method is proposed for exchange independent information in two wireless nodes using only XOR for network coding (Wu et al., 2005).

# Network Coding Fundamental Theory

## Definition of Graph

The network *G* = (*V*, *E*, *C*) is a Directed Acyclic Graph (DAG), *V* is the set of vectors and *E* is the set of edges. For each directed edge <*i,j*>∊*E* there is a positive integer capability *Ci,j*∊*C*. The *source node*, in which information is generated, is denoted by *s* and the *target node*, also called *sink node* or *receiver node*, is denoted by *t*. The set of all the incoming edges of one node *u* (*u∊V*)is *in*(*u*) and all the outgoing edges of node *u* is *out*(*u*).

## Min-Cut and Max-Flow

A flow *F* of the network is a set of positive values for each edge <*i*,*j*> which satisfies:

0≤ *F*i,j ≤ *C*i,j, <*i*,*j*>∊*E* (1)

For every node except *s* and *t*, the incoming flow is same as the outgoing flow as described below:

(2)

The outgoing flow of the source node *s* is same as the incoming flow of the sink node *t*. And this value is defined as the total value of the flow *F*.

(3)

A max-flow is a flow F with the maximum total value |F| than any other flow over the network.

The *cut set* *U* is a subset of node set *V* such that the source node *s*∊*U* and the target node *t*∉*U*. The edges across the cut set *U* could be represented as:

EU={<i,j>∊E : <i,j>∊out(u)∩ in(v), u∊U, v∊V} (4)

The capacity of the cut set *U* is the total capacity of edges in *EU*

(5)

Similarly as *max-flow*, the cut set *C* with minimum |*C|* in one network is called *min-cut*. Intuitively, the *min-cut* is the bottleneck between node *s* and node *t*, and any *max-flow* could not exceed the *min-cut*.

There is an example network shown in Figure 5(*a*), the source node is *s* and the sink nodes are , the capacity Ci,j of each edge <*i*,*j*> is marked besides it.

7

3

2

3

3

3

1

4

3

2

4

2

2

2

3

*t*1

*t*2

*s*

(*a*)

(3,7)

(2,3)

(2,2)

(2,3)

(0,3)

(0,3)

(1,1)

(1,4)

(3,3)

(2,2)

(2,4)

(0,2)

(2,2)

(0,2)

(0,3)

t1

*s*

t2

(*b*)

(2,7)

(3,3)

(2,2)

(2,3)

(1,3)

(1,3)

(0,1)

(0,4)

(0,3)

(0,2)

(2,4)

(2,2)

(0,2)

(2,2)

(3,3)

*t2*

*t1*

*s*

(*c*)

1. Two sink nodes network with two max-flow examples

The max-flow *F*1 from *s* to *t*1 could be worked out with any one of the algorithms discussed in Section III (A). The result is shown in Figure 5 1 (*b*). The actual flow values and the capacity limitation are indicated as a pair in the form of “(*F*i,j ,*C*i,j)”. The edges with no flow (*F*i,j=0) are marked with dash lines in the graph. The flows are: 1 unit along path 0-1-3-6-9; 2 units along path 0-1-4-6-9, 2 units along path 0-2-4-7-9. Total flow value is 5.

Similarly, the max-flow *F*2 from *s* to *t*2 is shown in Figure 5(*c*). The flows are: 1 unit along path 0-2-5-8-10; 2 units along path 0-2-4-8-10, 2 units along path 0-1-4-7-10. Total flow value is 5. All the edges shared by both flow are marked with bold lines.

## Min-Cut Max-Flow Theorem

Theorem 1 (Menger, 1927): (Max-flow Min-cut Theorem) the max-flow of a certain network *G* is exactly same as the min-cut, i.e.

(6)

This theorem indicates that the max-flow could achieve the min-cut bottleneck. Furthermore, one possible transportation scheme is obtained if the max-flow is worked out for one source and one sink network. It is intuitive that the max-flow is the upper bound of the capability of transportation from *s* to *t*.

In addition, in most of network coding research, the capacity of each edge is simplified to 1, or so called *unit capacity* link.

## Multicast Problem

The multicast information flow problems is where most research has taken place.

In the multicast problem, there is only one source node and multiple sink nodes. All messages are available at the source node *s*, each target node *t*i requires all the messages.

## Max-flow Bound Theorem

Define the max-flow from *s* to every sink node *ti* is (1≤*i*≤*l*).

|  |  |  |  |
| --- | --- | --- | --- |
| Problem | Upper bound of Rate | Routing Scheme | Algorithm Time Complexity |
| Unicast |  | Max Flow | Polynomial |
| Broadcast |  | Spanning Trees | Polynomial |
| Multicast |  | Steiner Trees | NP |

1. Routing scheme rate and algorithm for uni/broadcast & multicast

In the unicast case, there is one source node *s*, and only one sink node *t*, the upper bound capacity is. Standard maximum flow algorithm could be applied and routing scheme is obtained at the same time.

In the broadcast scenario, there is one source *s* and all the other nodes in *V* are sink nodes. Clearly, the upper bound capacity of broadcast is limited by the sink node *t* with the minimal. It was proved that this capacity is achievable by spanning trees (Edmonds, 1973). The spanning trees could be found in polynomial time and packets could be routed over them to achieve the upper bound capacity.

In the multicast scenario, the *l* sink nodes *t1*, *t2*, … *tl* are all belongs to a set . Similarly, the upper bound of capacity is . Unlike the broadcast problem, the Steiner tree for multicast is a NP-hard problem which could not be resolve in polynomial time (Jain et al., 2003). Thus, the upper bound is not achievable until the birth of network coding. It was proved that the upper bound of multicast capacity of the network is the minimum one among the max-flows to different multicast sink nodes.

Theorem 2 (Ahlswede et al., 2000): For a network with capacity constraints, the upper bound of the multicast capacity *w* is

(7)

In short, “network coding makes it possible to achieve maximum throughput given by max-flow min-cut theorem, which might not be achieved if only routing is allowed.” (Sun, 2005)

This theorem is also known as the network coding main theorem.

This upper bound capacity can be achieved by Linear Network Coding (LNC) or Random Network Coding (RNC) as discussed below.

## Linear Network Coding

Linear Network Coding is a linear combination of packets received at one node into one or more outgoing packets (Li et al., 2003). Assume that there is *L* bits in each packet, every *s* sequential bits of a packet forms one symbol over the field F2­s with each packet becomes a vector of *L/s* symbols. Addition and multiplication could be performed over the field F2­s. The result encoded packets are in the same length of *L*.

Assume that a number of original packets *M*1, ... *M*n are generated at the source node. In linear network coding, a sequence of coefficients *g*1, …, *g*n is picked up from F2­s. The encoded packet is

(8)

However, one encoded packet does not carry all the information of the source packets. At the decoding node, sufficient number *m* (*m*≥*n*) of encoded packets and coefficients pairs (*g*1, *X*1), …, (*g*m, *X*m) are required in order to recovery the original packets *M*1, …, *M*n. In other words, it is required to solve the *m* equations and *n* unknowns.

(9)

In order to solve these equations, the coefficients vectors should be linear independent, i.e. the *n*×*m* matrix should be full rank.

(10)

The original packets *M*1, …, *M*n could be works out with the following formula:

*M*=*G*-1×*X* (11)

## Random Linear Coding

Random Linear Coding provides a distributed manner for network coding.

It was shown that if the coefficients are picked up randomly from a large enough range such as 216, the matrix **G** is full rank with the probability very close to 1 (Ho et al., 2004).

Random network coding is independent from the network topology. So it could be constructed even if the network topology is unknown.

In addition, exponential time complexity algorithm, polynomial-time algorithm and greedy algorithm could be used to construct the coding coefficients.

## Routing for Network Coding

### Coding at Merging Node

One node is called the merging node if there is more than one incoming links. As network coding requires more than one packet to operate, the merging node is a requirement for the coding operation. In most network coding research, the coding operation takes place at all the merging nodes.

### Multicast Coding Routing

Two methods can be used to determine the coding nodes and path for a single source multiple terminates graph. Take the butterfly network for example, as the standard max flow algorithm is designed for one source and one terminate, the max flow for the two terminates are worked out separately. A coding sub graph is generated based on the two flows and the finally coding scheme is determined (Lun et al., 2005).

Another method modifies the labelling algorithm by adding the target node to the mark so that different targets can be worked out by one max flow algorithm. The time complexity is increased to O (*tmn*2) where *t* is the number of the target nodes (Tao et al., 2008).

T1

T2

V1

V2

V3

V4

*(a)*

*(b)*

S

T1

T2

V1

V2

V3

V4

S

1. Butterfly Network Routing Examples

As shown in Figure 6, the two maximum flows are listed in (a) and (b) respectively. For node V3, there are two incoming path, and both max flow shares the edge between V3 and V4, hence V3 should be the encoding node for network coding.

# Requirement Analysis and Specification

## Problem Description

The network is represented by directed acyclic graph *G* = (*V*, *E*, *C*) in Section 3. In the multicast scenario, one source node *s*∈*V* transmits *w* data blocks simultaneously to sinks nodes in a set *T*⊂*V*, where *w* is the maximum multicast capacity defined in Theorem 2. The problem is to find a transmission scheme with network coding that all the data blocks is received by all the sink nodes. The number of data blocks at each link is less than or equal to the capacity constrain.

As network coding is a novel research field, there are few complete algorithms to construct complete coding and routing schemes. Although there is efficient algorithm for linear network coding, the large number of coding nodes requires considerable computational costs. Generic Algorithm is used to optimal the number of coding node but it is very time consuming and complicated.

## Algorithm Design Objectives

This project is aimed to design an algorithm to solve this problem. For a given network, a complete coding and routing scheme will be constructed as a result. Therefore, the objectives of this project are as follows.

* The maximum multicast rate of a network should be achieved.
* There is no unit capacity limitation on each edge.
* The number of coding nodes should be minimised.
* If there is no coding node required for a network, the algorithm will provide a transmission scheme with traditional forward and multicast mechanism.

## Implementation Specification

The proposed algorithm above will be implemented and validated.

As this project is major in algorithm design, the program is focus on implementation of the algorithm rather than human computer interaction, therefore a light weight script programming language could be chosen as development language. The language should provide inherent data structure for set operation and data lists and matrixes.

For large amount of test cases, input and output of the program would be data files. For human readability, plain text is used as the input data file format and the outputs are visualized in graphs.

The desired functions of the program including:

* Load the network from input file and represent it in the memory
* Work out max-flow for obtain the maximum multicast rate
* Work out the coding and routing scheme with the designed algorithm
* Visualise the input network, coding and routing scheme
* Visualise the intermediate steps such as merged max-flow, target sets
* Generate random graph for testing and validation

## Network Example

One example network with one source and three sink nodes 5, 6 and 9 is shown in Figure 7(*a*) (Fragouli, 2007), all the edge has an unit capacity. However, there is a cycle 6-7-8-6. In order to remove the cycle, one new node 10 is introduced in Figure 7(*b*), edge <8,6> is removed and new edge <6, 10> and <8, 10> are introduced and node 10 is regarded as the sink node *t*1 instead of node 6. The max flows from *s* to sink nodes *t*1, *t*2 and *t*3 are all 2. One network coding scheme that achieve the maximum multicast capacity is demonstrated in Figure 7(*c*). According to the definitions, the coding nodes are node 2 and 7, and the multicast nodes are node 1, 3, 4 and 8.

*t*2

*t*3

*t*1

*s*

2

7

*t*2

*t*3

*s*

*a*

*b*

*a*

*b*

*a*

*a*

*b*

*b*

*b*

*b*

*a+b*

*a+b*

*a+b*

*a+b*

(*b*)

(*c*)

*t*1

*a*

*t*1

*s*

(*a*)

1. Unit capacity three sink node network coding scheme

From the two examples above, it is shown that actually the coding node have followed the pattern of the butterfly network.

# Algorithm Design for the Coding - Routing Scheme

First of all, multicast flow graph is constructed by merging all the max-flow paths to every sink node. Second, all the data blocks are packed into packets along with its destination node index. Finally, these packets are dispensed from source node to sink nodes through forward, multicast or coding operations.

## Construction of Multicast Flow Graph

### Max-Flow Algorithms

Max flow of one graph can be found out through two types of algorithm: augmenting path methods or pre-flow-push methods.

Augmenting path method, also known as the Ford Fulkerson method, is based on the idea of finding out one path that could enhance the current flow, until no path could be found. Let |E| = *n*, |V|=*m*, and assume maximum edge capability is *u*. The general labelling algorithm is finding any possible path with the time complexity of O (*nmu*). The capacity scaling algorithm limited the path to be the one with maximum increasing improvement and reduces the time complexity to O (*nm* log *u*). Another shortest augmenting path algorithm - Edmonds Karp algorithm aims at finding out the path with the smallest number of nodes, e.g. the Board First Search (BFS). The time complexity is O (*nm*2).

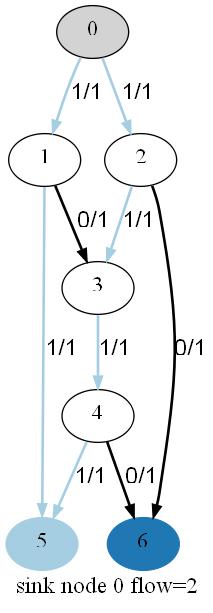
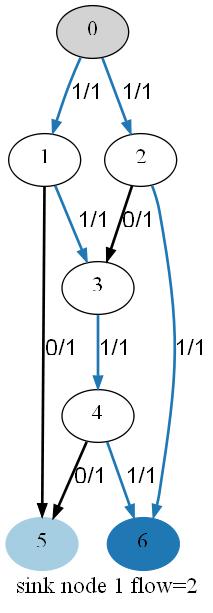
### Labelling Algorithm

The high level of the iterative Ford-Fulkerson-Method is described below:

1. procedure labelling(net, s, t)
2. initialize *flow* to 0
3. while there exists an augmenting path *p*
4. augment *flow* along *p*

The pseudo code of the labelling algorithm to work out the max-flow from node *s* to node *t* is described below:

1. *net*[i][j]is the capacity of edge <i,j>
2. *flow*[i][j] is the flow value of each edge <i,j>
3. *res*[i][j] is the residual capacity for edge <i,j>
4. *pred*[i]ispredecessor of node *i* in the augmenting path
5. *list* is the set of nodes to be visited while seeking the augmenting path
6. procedure labelling(net, s, t)
7. begin
8. label node *t*
9. while *t* is labelled do
10. begin
11. # Find augmenting path
12. unlabel all nodes
13. set *pred*[*i*] ← 0 for every node *i*∈*V*
14. label node *s*
15. *list* ← {*s*}
16. while list≠∅ or *t* is unlabeled do
17. begin
18. remove a node *i* from *list*
19. for each edge <*i*,*j*> in the residual network do
20. *res*[*i*][*j*] = *net*[*i*][*j*] – *flow*[*i*][*j*] + *flow*[*j*][*i*]
21. if *res*[i][j]>0 and node *j* is unlabeled then
22. begin
23. *pred*[*j*] ← *i*
24. label node *j*
25. append j to list
26. end
27. end
28. # Augment the path
29. if t is labelled then
30. begin
31. use *pred* to trace back from *t* to *s* to obtain an augmenting path *P*
32. *delta* := min {res[i][j] : <i,j>∈P}
33. augment *delta* units of flow along P
34. end
35. end
36. end

(*a*) (*b*)

1. Max-flows to each sink node of butterfly network

For example, the two max-flows to sink node 5 and sink node 6 are shown in Figure 8(*a*) and (*b*), respectively. In this figure, flow values *Fi,j* and capacity values *Ci,j* are separated by slash “/” and placed beside each edge. The edges with non zero flow are marked with same colour with the sink node. Both sink nodes have the same maximum flow value 2. The two paths to sink node 5 are 0-1-5 and 0-2-3-4-5. Similarly, the two paths to sink node 6 are 0-1-3-4-6 and 0-2-6.

### Merging Max-Flow for Obtaining Multicast Flow Graph

The unit capacity of each link is a common assumption in most network coding approaches. Larger capacity is substituted with multiple parallel links in the same direction between two nodes. However, link capacity in real networks is always a quantity in a large number instead of 1. The approaches based on unit capacity encounter a problem to deal with a large number of links with unit capacity. Hence the computational complexity is increased dramatically. The proposed approach here avoids this problem. Since links are not unit capacity, the new definitions are proposed as follows.

Definition 1: The *merged max flow* to all the sink nodes is defined as



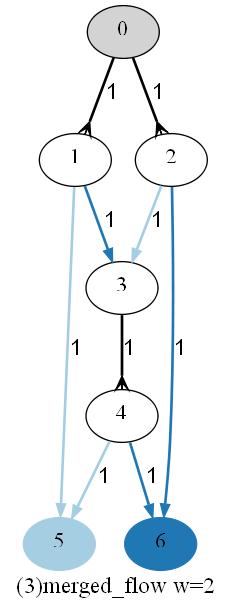
That is to say, for every edge, the *merged max flow* is the maximal one of the single max flows overlapping on the edge. The value of the *merged max flow* is used as the new limitation of the edge capacity.

Definition 2: The *overlapped edge* is the edge used in two or more max-flows to different sink nodes, i.e. the edge <*i*, *j*> satisfies

Fti,j >0 ∧ Fki,j>0 (1≤*t*≤*l ,* 1≤*k*≤*l, k≠t, <i,j>*∊*E*)

For the example in Figure 8, the combined maximum flow for the two max-flows is shown in Figure 9. The overlapped edges are marked with bold lines. The numbers beside the edges are the merged capacity limitations.

According to Theorem 2 in Section 3, the upper bound multicast transportation capacity is the minimum one of the two single max-flows. For this example, it is *min*(2,2)=2, i.e. there are 2 different packets can be broadcasted from *s* to *t*1 and *t*2 at one time unit if network coding is used. At the end of this algorithm a network coding scheme under this capacity limitation will be constructed. The merged flow graph simplifies the network topology. Only the edges and nodes in the merged max-flows will be used to construct the coding and routing scheme,



1. Merged max flows for butterfly network

### The Target Set of an Edge

Definition 3: The *target set* *of edge <i,j>* is denoted as *Di,j,* for any sink node *t* in *Di,j*, the edge <*i*,*j*> is on the max-flow from *s* to *t*. The sink nodes in the target set is represented with its index number. The formal definition is as following:

  
*T* is the set of all sink nodes

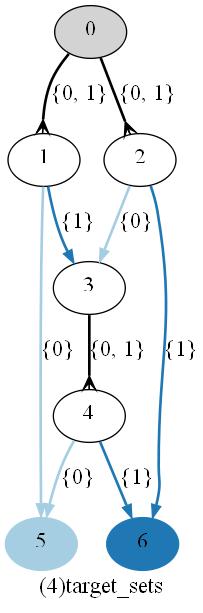
*l* is the number of sink nodes

is the flow value of edge <*i,j*> to sink node *td*

For unit capacity, as ,  is equivalent to 



For example, the target set *D* for butterfly network is shown in Figure 10. It could be derived from Figure 8. There are two sink node 5 (*index*=0) and sink node 6 (*index*=1). Edge <0, 1> is shared by both max-flows, so the target set *D*0,1 is {0,1}; Edge <1, 3> is on the max-flow of node 6 only, so the target set *D*1,3 is {1}. The other edges could be done by parity of reasoning.



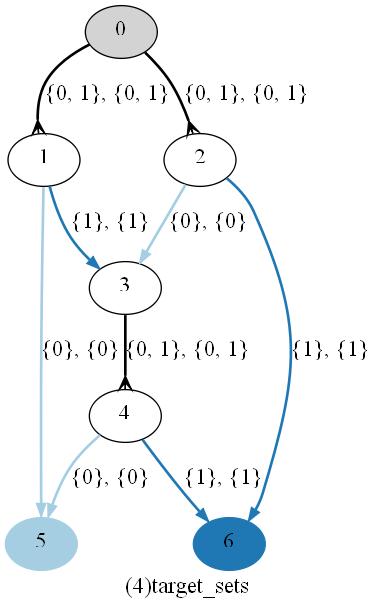
1. Target sets for butterfly network

For multiple capacity edges, there are multiple target sets at each edge <i,j>. The number of target sets is the flow value of the merged flow . Each target set is defined as follows:



Intuitively, the multiple capacity edge is treated as several parallel edges, and each edge is assigned a target set.

For example, if the capacity of each edge at the butterfly network is doubled. There are two target sets for each edge as shown in Figure 11.



1. Target set for double capacity edge of butterfly network

## Packet Representation and Operations

### Representation of Packet

The task of multicast via a network can be regarded as a performance in which all the information packets should be transferred to multiple sink nodes. For simplicity of implementation, suppose that a packet is composed of two parts: target set and data block. Target set of a packet contains a set sink nodes where the packet should be sent to. The data block is a information unit carried by the packet. A packet *p* is represented as the format shown as follows

*p* = (*data block list, target set*)

The *data block* is a fixed length data section. For simplicity in algorithm design, one data blocksis denoted with one letter from “*a”* to “*z”*. The multiple data blocks in a *data block list* are linear combined (this refers to network coding) and the length is equal to one data block. A packet can carry a single data block or a data block list (coded data block) and single or multiple destinations in its target set. The data block can be coded or decoded, and the target sets can be split or merged.

The element in *target set of a packet* is the index number of sink nodes, e.g. 0, 1, 2, …, *l*. For example, one packet of data block *a* sending to the first and second sink node is represented as “(*a*, {0, 1})”.

The target set of packet *p* is represent as *D*(*p*) and the data block list of packet *p* is represented as *B*(*p*).

At the beginning, there are *w* data blocks at the source node (*w* is the maximum multicast capacity of the network). Every data block should be transferred to every sink node. So there are *w* packets, every packet consists of a unique data block and a full target set {*i*: 0<*i*<*l*-1}.

For example, for a network with *l* = 3 and *w* = 2, all the original packets are listed below:

(*a*, {0, 1, 2}), (*b*, {0, 1, 2})

There are two operations on the packets: split and coding.

### Split Target Set of Packet

The *split* operation divides the target set of one packet according to the specified subset *ts*, and the data block list is unchanged. The pseudo code is shown below.

1. **procedure** split(packet *p*, target set *ts*)
2. **begin**
3. new packet *q* ← (B(*p*), *ts*)
4. new packet *r* ← (B(*p*), D(*p*) – *ts*)
5. return *q*, *r*
6. **end**

For example, packet “(*a*, {0, 1, 2})” could be split to “(*a*, {0})” and “(*a*, {1, 2})” with a specified subset {0}.

### Coding Data Blocks of Two Packets

The *coding* operation joins the data blocks of two packets together. The target sets are union at the same time. The pseudo code is shown below:

1. **procedure** coding(packet p, packet q)
2. **begin**
3. p ← (B(p) + B(q), D(p) ∪ D(q) )
4. return p
5. **end**

For example, coding two packets “(*a*,{0})” and “(*b*,{1})” will result in “([*a,b*],{0,1})”. If *xor* operation is applied on the data blocks, the list “[*a*, *b*]” corresponds to “*a xor b*”.

## Algorithm for Construction of Coding and Routing Scheme

At the beginning, all the packets are generated at the source node. The idea is to transmit all the packets along the max-flow paths to its target nodes.

In order to avoid back-trace, all the nodes are visited only once by the topological ordering. When visiting one node, all the packets at that node are transferred to its successor nodes through an outgoing edge by using three operations one by one: forward operation, multicast operation and coding operation. Each operation might process partial or all the packets at that node. After the forward operation, the multicast operation is performed, only if there are any packets left. If there are still any packets after multicast operation, coding operation is applied to handle all the remaining packets.

### Node Arrangement in Topological Order

For directed acyclic graph (DAG), topological ordering is a linear ordering of all the nodes. All the nodes in the list do not have outbound edges to any node in front of it. Obviously, the source node could be the first node. Thereby, in this upstream to downstream ordering, the coding scheme could be constructed to ensure that all the packets are passed onto the sink nodes eventually.

The topological ordering could be obtained by the topological sorting algorithm (Kahn, 1962). The algorithm recursively chooses a node which has no incoming edges, puts it the output list and removes all the outgoing links of that node. Such a node will always exist unless the graph has a cyclic. The time complexity is linear as all the nodes or edges would be accessed once, O(|V|+|E|).

The pseudo code of the algorithm is described below:

1. L is the list that will contain the sorted elements
2. S is set of all nodes with no incoming edges
3. L ← ∅
4. *S* ← {*s*}
5. **while** S is non-empty do
6. pop a node i from S
7. insert i into L
8. **for** each node j with an edge e from i to j do
9. remove edge e from the graph
10. if j has no other incoming edges then
11. insert j into S
12. **if** graph has edges then
13. output error message (graph has at least one cycle)
14. **else**
15. output message (proposed topologically sorted order: L)

For example, the topological ordering for butterfly network is 0, 1, 2, 3, 4, 5, 6.

Another topological sort algorithm based on Deep First Search is discussed in.

### Forward Operation

Using forward operation, the packets are transferred to the outgoing edge with the exactly same target set.

Usually forward operation can transfer all the packets at the nodes which the sum of incoming flows is same as the sum of outgoing flows on the merged flow graph.

The pseudo code of forward operation is straightforward:

1. **procedure** forward\_operation(node *i*)
2. **begin**
3. **for** each packet p in node i
4. **for** each outgoing edge <i,j>
5. if packet target set equals to edge target set then
6. send packet p to node j through edge <i,j>
7. **end**

For example in Figure 12, at the source node of butterfly network, the packet (a, {0,1}) is transferred to edge <0, 1> with target set {0, 1} and packet (b, {0,1}) is transferred to edge <0, 2> with target set {0,1}.

{0, 1}

{0, 1}

(*a,*{0, 1})

(*b,*{0, 1})

(*b,*{0, 1})

(*a,*{0, 1})

1. Forward operation at source node

### Multicast Operation

After the forward operation, all the packets with fully matched target set are allocated. So the remaining packets may be only partially matched with the target set for some edges. By enumerating all the pairs of packet and edge, the pair with maximum intersection of the target set is chosen. Then the packet is split into two packets, with one packet to be sent along the edge and the other remained at the node. This operation is repeated until there is no matching pair of packet and edge pair.

The pseudo code of iterative multicast operation is shown below:

1. merged[i][j] is the merged flow value of edge<i,j>
2. fill[i][j] is the currently filled flow value of edge <i,j>
3. **procedure** multicast\_operation(node *i*)
4. **begin**
5. **while** there is any change
6. **begin**
7. **for** each successor node j’
8. **if** fill[i][j’] < merged[i][j’]
9. **for** each edge slot k’ **if** k’ is not occupied
10. **for** each packet *p’* in node *i*
11. *overlap* = D(p’) ∩ Dk’i,j’
12. if *overlap* > *max\_overlap* then
13. *p* ← *p*’
14. *j* ← *j*’
15. k ← k’
16. max\_overlap ← overlap
17. found ← true
18. packets *q*, *r* ← split packet *p* with target set *overlap*
19. send packet *q* to node *j* through edge <*i*,*j*>
20. update target set at edge <i,j>
21. increase fill[i][j]
22. keep packet *r* in node *i*
23. **end**
24. **end**

As shown in Figure 13(*a*), there is a multicast operation at the node 1 of butterfly network. The best matched edge for packet (*a*,{0, 1}) is <1,5>, the intersection of target sets is {0}. Hence, packet (*a*,{0, 1}) is split into two packets (*a*,{0}) and (*a*,{1}), then (*a*,{0}) is transferred to the outgoing link <1, 5> and (*a*,{1}) is retained at the node 1. Thereafter, packet (*a*, {1}) is found to be matched with edge <1, 3> with the same target set {1}, hence it is sent to node 3. In the coding scheme, this multicast operation is illustrated in Figure 13(*b*).

In the realistic scene, the router at node 1 replicates the data block *a* into two packets with different target set and send them to two outgoing links.

(*a,*{0, 1})

{1}

{0}

(*a,*{0})

(*a,*{1})

(*a*)

*a*

*a*

(*b*)

1. Multicast operation example

### Coding Operation

The coding operation resolves the remaining packets that could not be handled by both forward and multicast operations. Similar to the multicast operation, the coding operation iteratively find the best matched target set between packets and outgoing links. The difference is that coding operation encodes two data blocks into one data block list to make a new packet. Two packets become one packet just take one unit of the flow capacity.

The pseudo code of coding operation is described below.

1. **procedure** coding\_operation(node *i*)
2. **begin**
3. found ← true
4. **while** found **do**
5. **begin**
6. found ← false
7. max\_overlap ← ∅
8. **for** each successor node j’
9. **for** each slot k’ on outgoing edge <i,j’>
10. **for** each packet p’ in node i
11. *overlap* = D(p’) ∩ Dk’i,j’
12. if *overlap* > *max\_overlap* then
13. *p* ← *p*’
14. *j* ← *j*’
15. k ← k’
16. max\_overlap ← overlap
17. found ← true
18. packets *q* , *r* = split *p* according to *overlap*
19. coding *q* with the existing packet on slot k of edge <i,j>
20. keep packet *r* in node *i*
21. **end**
22. **end**

One example of coding operation in the butterfly network is shown in Figure 14(*a*). There are two packets (*a*,{1}) and (*b*,{0}) arriving at node 3. However, there is only one outgoing link with unit capacity and target set {0, 1}. Hence, two packets are required to be combined together. Firstly, packet (*a*, {1}) is sent to node 4 through <3, 4> by multicast operation. At this time, the D3,4={0,1}-{1}={0}. Next, by a coding operation, packet (*b*,{0}) is going to be coded with packet (*a*,{1}). The new data block list is [*a*,*b*] and the target set for new packet is {1}∪{0}={0,1}. If *xor* is used as coding operation, the coding scheme is shown in Figure 14(*b*).

In the realistic scene, the router at node 3 working out the *xor* result of the data block *a* and data block *b*, then sends the result to the only outgoing link <3, 4>.

3

{0}

{1}

(*a,*{1})

(*b,*{0})

(*a*)

([*a*,*b*]*,*{0, 1})

{0, 1}

3

*a*

(*b*)

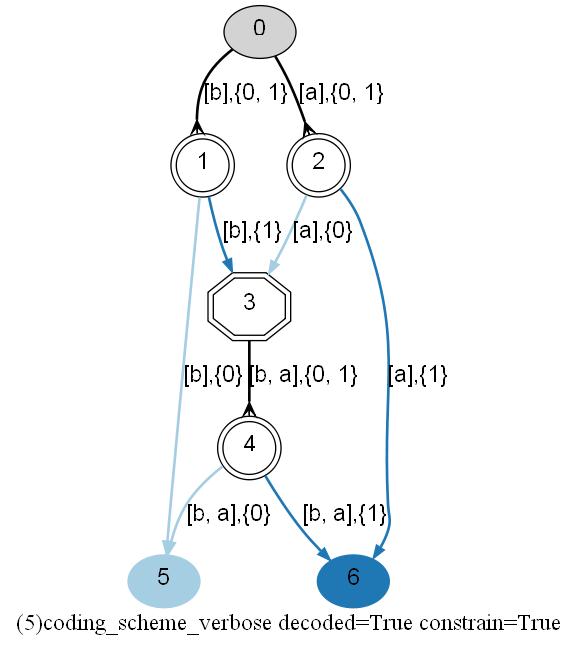
*b*

*a*⊕*b*

1. Coding operation example

### Coding and Routing Scheme Example

After performing the operations above to all the nodes in topological order, the coding scheme for butterfly network is constructed as shown in Figure 15.



1. Coding scheme with packet target set

### Identification of Node Types

Based on the operations in a node, all the nodes can be classified into three types according to the type of operations performed on the node. Therefore, node types are defined as follows:

Definition 4: A node is called *forward node* if only forward operation is used to transfer all the received packets at the node.

Definition 5: A node is called *multicast node* if at least one multicast operation should be performed to transfer all the received packets and not any coding operation is required to perform at the node.

Definition 6: A node is called *coding node* if at least one coding operation is required to transfer all the received packets at that node.

Based on these definitions, the actual coding nodes can be found using the proposed algorithm.

# Implementation of the Proposed Algorithm

## Development Environment

### Python Programming Language

Python is a dynamic interpreted language with elegant syntax and efficient high-level data structures. The Python interpreter and standard library are all open-source. There is a considerable amount of free third party modules programs and tools as well. Python can run on Windows, Linux/Unix and Mac OS X. Python have been used in scientific and numeric domain like bioinformatics and physics.

The benefits of Python are that source code is easy to read and very close to the pseudo code, projects are easy to maintain and there are wide selection of third party libraries and modules.

### Graph Visualization with Graphviz

Graphviz is a graph visualization software for representing structural diagrams of abstract graphs or networks. Graphivz is open source and has many applications in software engineering, database and web design as well as in networking.

The Graphviz program takes descriptions of graph in a text description language (Dot) and automatically designs the layout of diagrams. Features such as colours, fonts, line styles and node shapes can be customised.

Pydot is a python interface to Graphviz’s Dot language. It allows the creation of figures for directed graph from Python data structures. All the attributes of the Dot language are supported.

## Data Structures

The adjacency matrix is used to represent the graph in the final implementation. For a network with *n* nodes, one *n*×*n* matrix *net* is used to store the edges and capacity limitation. A edge is represented with capacity as follows.



A flow through edge <*i*,*j*> is represented as



A merged flow through edge <i,j> is represented as



The python data type “list” is used to store the matrix.

All the nodes are numbered from 0 to *n*-1.

The python code to enumerate all the edges of the network is

1. **for** i **in** range(n):
2. **for** j **in** range(n):
3. **if** net[i][j]:
4. do something for edge <i,j>

The code to enumerate all the incoming edges of the node *i* is

1. # at node i
2. **for** j **in** range(n):
3. **if** net[j][i]:
4. # do something for edge <j,i>

Similarly, the code to enumerate all the outgoing edges of the node *i* is

1. # at node i
2. **for** j **in** range(n):
3. **if** net[i][j]:
4. # do something for edge <i,j>

One path is represented with an *n* element predecessor array *pred*. pred[i] which stores the previous node of node *i* in the path. The traverse of the path starts from the last node *t* until the first node *s*.

The python code is listed below:

1. i = t
2. **while** i != s:
3. j = pred[i]
4. # do something for edge <i,j>
5. i = j

## Data File Format

### Input File format

The first line contains four positive integers: *n* giving the number of nodes; *m* giving the number of edges; *s* giving the source node id and *l* giving the number of sink nodes.

The second line contains *l* integers, giving the id of sink nodes.

Then followed by *m* lines, each line contains three positive integers: *x* stands for the source node of edge; *y* stands for the end node of edge and *c* stands for the capacity of edge <*x*,*y*>.

One sample input file for butterfly network is listed below.

7 9 0 2

5 6

0 1 1

0 2 1

1 3 1

1 5 1

2 3 1

2 6 1

3 4 1

4 5 1

4 6 1

### Output File Format

Log files include all the intermediate matrixes, coding scheme results, warning and error messages.

Furthermore, there are six types of output images files:

1. Input network
2. Max-flow to each sink node (Figure 8)
3. Merged max-flow (Figure 9)
4. Target sets at each edge (Figure 10)
5. Coding scheme with data blocks and target sets(Figure 15)
6. Coding scheme with *xor* operation

In order to distinguish different flows to different sink nodes, “*paired12*” colour scheme provided by Graphviz is chosen as shown in Figure 16. Each sink node and the related max-flow are assigned an identical unique colour.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12* |

1. Colour scheme for different max-flows

The nodes are represented with unique shape to its type as well. As shown in Figure 17, the normal cycle shape stands for forward node; double cycle shape stands for multicast node; double octagon stands for coding node.

http://www.graphviz.org/doc/info/circle.gif

1. Node shapes of output images

As shown in Figure 18, two arrow shapes are selected: crow is used for overlapped flow to multiple sink nodes and normal arrow is used for single flow to single sink node.

|  |  |  |
| --- | --- | --- |
| "crow" | http://www.graphviz.org/doc/info/a_crow.gif | |
| "normal" | | http://www.graphviz.org/doc/info/a_normal.gif | |

1. Arrow shapes of output images

## Flow Chart of Program

The proposed approach is performed by a set of algorithms which are implemented in a program. The program loads the input file and constructs a coding and routing scheme for the network corresponding to the input file. The program also provides coding node number and link number as well as other statistics. The work flow of the program is shown in Figure 19.



1. Work Flow Chart of Implementation

## Code Examples

### Codes for Drawing Figures

The python code for drawing a single max flow is listed below. One directed Dot graph is initialized with pydot module in line 2. One edge is created by specifying two nodes’ name in line 8. From line 9 to 14, the properties of the edge are specified by the “set” methods. Finally, the edge is added to the graph in line 15. Similar, the nodes are added in line 17-23. The jpeg file is created by calling dot program in the end.

1. def draw\_flows(net, flow, flow\_index, n):
2. # initialize graph
3. graph = pydot.Dot(graph\_type='digraph')
4. # add edges
5. **for** i **in** range(n):
6. **for** j **in** range(n):
7. **if** net[i][j] > 0:
8. edge=pydot.Edge(str(i), str(j))
9. edge.set\_colorscheme(colorscheme)
10. edge.set\_style("bold")
11. edge.set\_fontname(edge\_font\_name)
12. edge.set\_label(str(flow[i][j]))
13. if flow[i][j] > 0:
14. edge.set\_color(str(flow\_index+1))
15. graph.add\_edge(edge)
16. # Add nodes
17. **for** i **in** range(n):
18. node = pydot.Node(str(i))
19. node.set\_shape("doublecircle")
20. node.set\_style("filled")
21. node.set\_colorscheme(colorscheme)
22. node.set\_color(str(T.index(i)+1))
23. graph.add\_node(node)
24. graph.write\_jpeg('output.jpg', prog='dot')

### Code for Topological Sorting

In the implementation of the topological sort, there is one new array *D* introduced to maintain the in degree for each node. Initially, *D*[*i*] is the total number of incoming edges of node *i*. During the sort progress, when edge *<i,j*> is removed from the graph, the in degree for node *j* i.e. *D*[*j*] is to deduct by one. To check if one node *i* has no incoming edge, check if *D*[*i*] equals to 0, e.g. line 21.

1. def topologicial\_sorting(graph):
2. global s, n
3. # Empty list that will contain the sorted elements
4. L = []
5. # Set of all nodes with no incoming edges
6. S =[s]
7. # In Degree of each node
8. D = [0 for i in range(n)]
9. for i in range(n):
10. for j in range(n):
11. if graph[i][j]:
12. D[j] += 1
13. # Topologicial sort
14. while S:
15. i = S.pop(0)
16. L.append(i)
17. for j in range(n):
18. if graph[i][j]:
19. D[j] -= 1
20. # if there is no incoming edge of node j
21. if D[j]==0:
22. S.append(j)
23. if max(D):
24. print "Warning: this is CYCLIC graph"
25. exit()
26. return L

# Testing and Experimental Results

## Test Cases for Max-Flow Module

The max flow program works out the max flow value from one source node to one sink node. It reads the network from text files and writes the output to another text file. It was tested using 30 handmade test cases. The format of input and output files have described as follows. The outputs of the program have exactly matched the standard output files.

The first line of input file contains four positive integers separated by space: *n m s t*

*n* is the number of nodes; *m* is the number of edges; *s* is the source node and *t* is the sink node.

Then followed by *m* lines, each line contains three positive integers: *x y c*

*x* is the source node of edge; *y* is the end node of edge and *c* is the capacity of edge <*x*,*y*>.

The first line of output file is one integer number of the maximum flow value.

Each of the following line represents one flow in the format of “(*x*, *y*) *f*”, it means that the flow value is *f* at edge <*x*, *y*>.

The test cases are listed below.

Test Case 1

|  |  |
| --- | --- |
| Input File | Output File |
| 4 5 0 3  0 2 4  0 1 2  1 2 3  1 3 1  2 3 5 | 6  ( 0 , 1 ) 2  ( 0 , 2 ) 4  ( 1 , 2 ) 1  ( 1 , 3 ) 1  ( 2 , 3 ) 5 |

Test Case 2

|  |  |
| --- | --- |
| Input File | Output File |
| 8 12 0 7  0 1 7  0 5 2  1 2 3  1 6 3  2 3 3  2 7 3  3 0 3  3 4 4  4 7 4  5 4 3  5 6 1  6 7 5 | 8  ( 0 , 1 ) 6  ( 0 , 5 ) 2  ( 1 , 2 ) 3  ( 1 , 6 ) 3  ( 2 , 7 ) 3  ( 4 , 7 ) 2  ( 5 , 4 ) 2  ( 6 , 7 ) 3 |

Test Case 3

|  |  |
| --- | --- |
| Input File | Output File |
| 5 8 0 4  0 1 5  0 2 7  0 3 3  1 4 6  2 1 2  2 4 5  3 2 4  3 4 8 | 14  ( 0 , 1 ) 5  ( 0 , 2 ) 6  ( 0 , 3 ) 3  ( 1 , 4 ) 6  ( 2 , 1 ) 1  ( 2 , 4 ) 5  ( 3 , 4 ) 3 |

Test Case 4

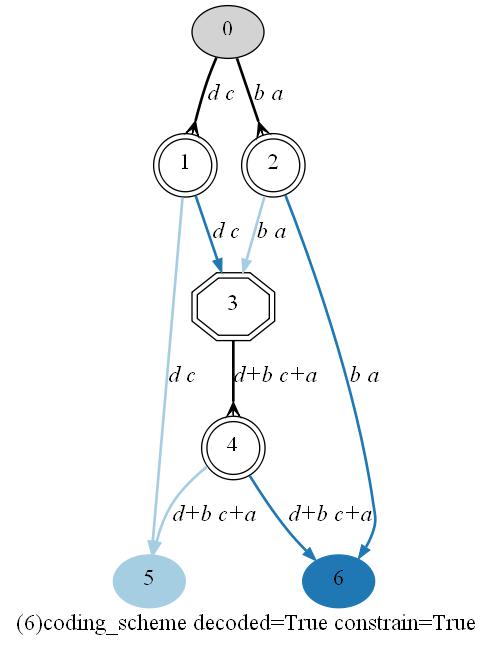
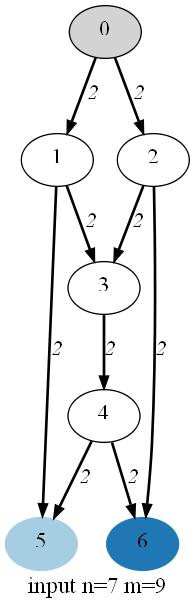
|  |  |
| --- | --- |
| Input File | Output File |
| 12 18 0 11  0 1 7  0 2 3  1 3 1  1 4 2  2 3 3  2 4 3  2 5 3  3 6 4  4 6 2  4 7 4  4 8 2  5 8 3  6 9 3  7 9 2  7 10 2  8 10 3  9 11 4  10 11 3 | 6  ( 0 , 1 ) 3  ( 0 , 2 ) 3  ( 1 , 3 ) 1  ( 1 , 4 ) 2  ( 2 , 4 ) 3  ( 3 , 6 ) 1  ( 4 , 6 ) 2  ( 4 , 7 ) 3  ( 6 , 9 ) 3  ( 7 , 9 ) 1  ( 7 , 10 ) 2  ( 9 , 11 ) 4  ( 10 , 11 ) 2 |

## Test Cases for Proposed Algorithm

### Typical Test Cases

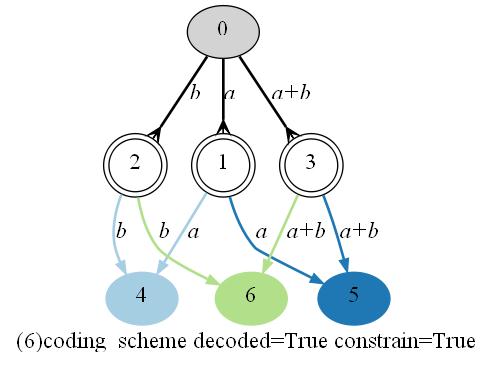
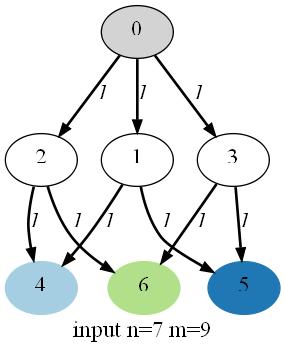
Thirty networks were created as test cases for the proposed algorithm (except for the butterfly network example). All the cases have obtained optimal coding and routing schemes successfully by the our algorithm. Six typical networks among them are listed below and some comments are given for the obtained coding and routing scheme.

As shown in Figure 20, the topology on the left is the same as the butterfly network, however, all the capacity of the edges are doubled. The coding scheme is shown on the right. There are two coded data blocks “*d*+*b*” and “*c*+*a*”, they could be decoded with single data block “*d*” and “*c*” at node 5, or a single data block “*a*” and “*b*” at node 6.



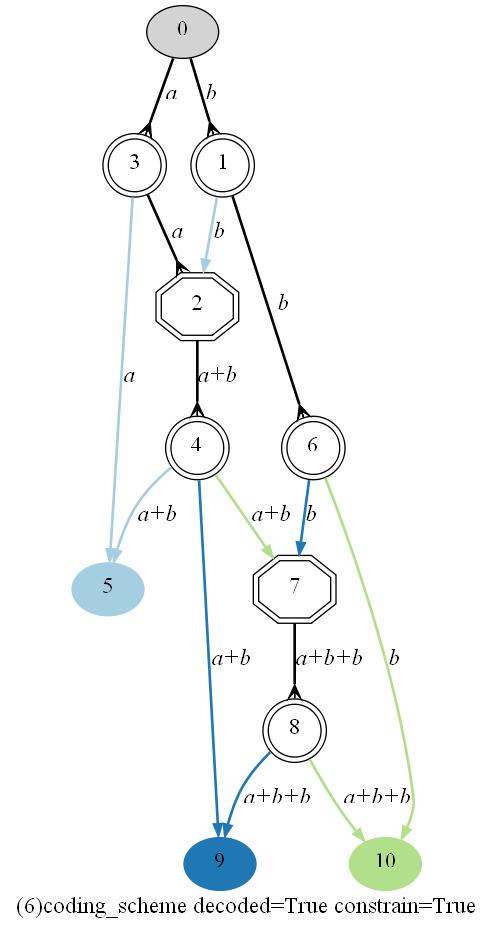
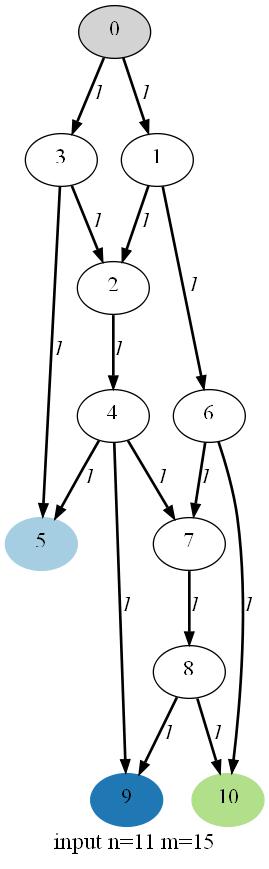
1. Double capacity butterfly network test case

As shown in Figure 21, the source node is acting as a coding node. Coded data block “*a*+*b*” is then multicast by node 3 and received by sink node 5 and node 6. Both sink nodes could decode it with another single data block “*a*” or “*b*”.



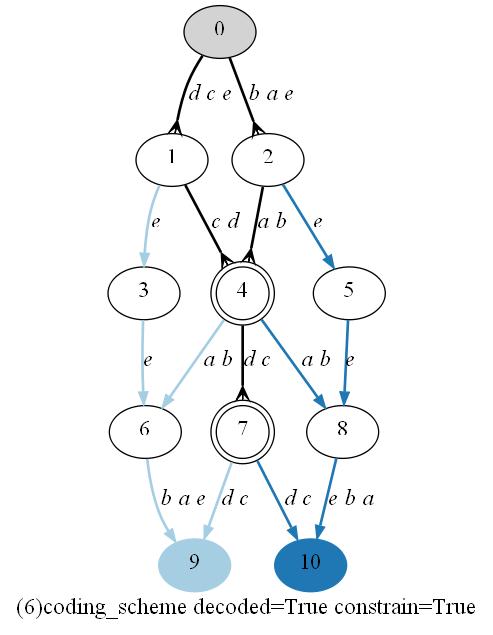
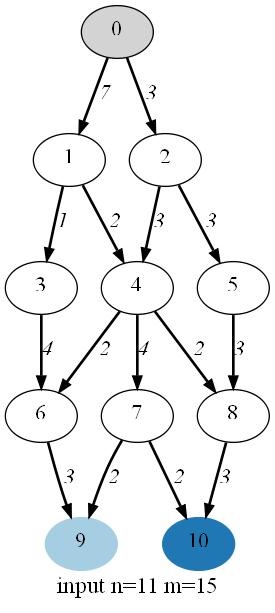
1. Coding at source node test case

As shown in , the coded data block might be coded again. The coded data block *a*+*b* is coded to *a*+*b*+*b* at node 7. Actually, *a*+*b*+*b* is equals to *a*. Node 9 could decode *a*+*b* with *a* and node 10 receives both *a* and *b*.



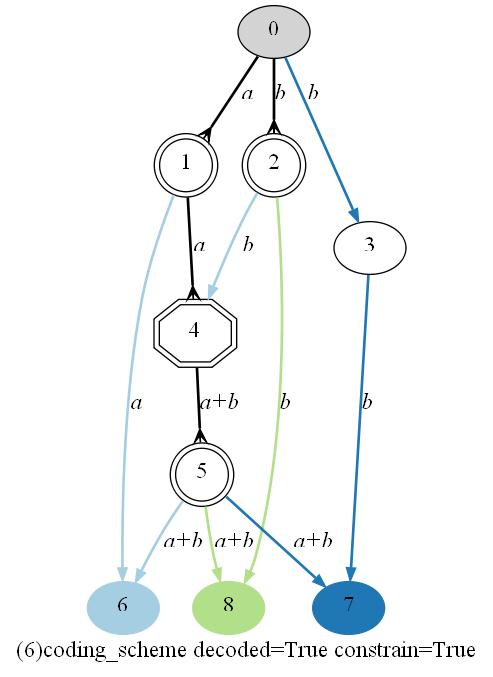
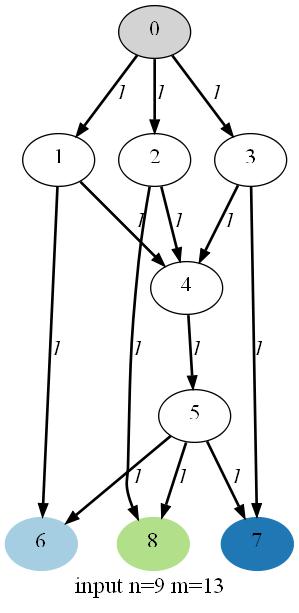
1. Two coding nodes test case

For some network, network coding is not necessary to achieve the maximum multicast capacity. As shown in Figure 23, the maximum multicast capacity 5 is achieved by forward and multicast operation with no coding operation.



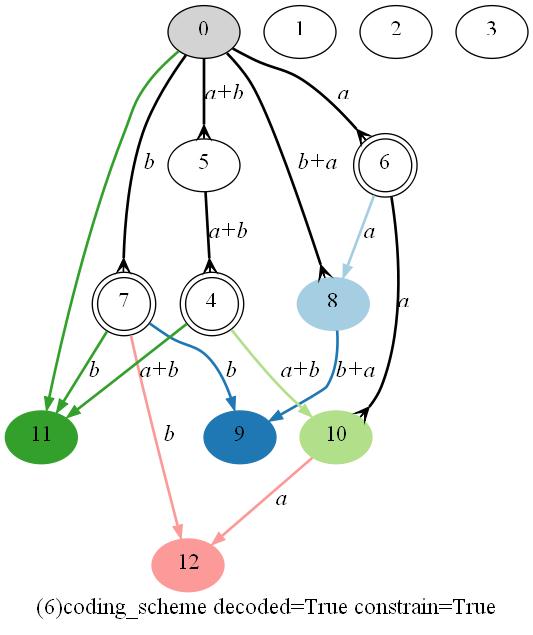
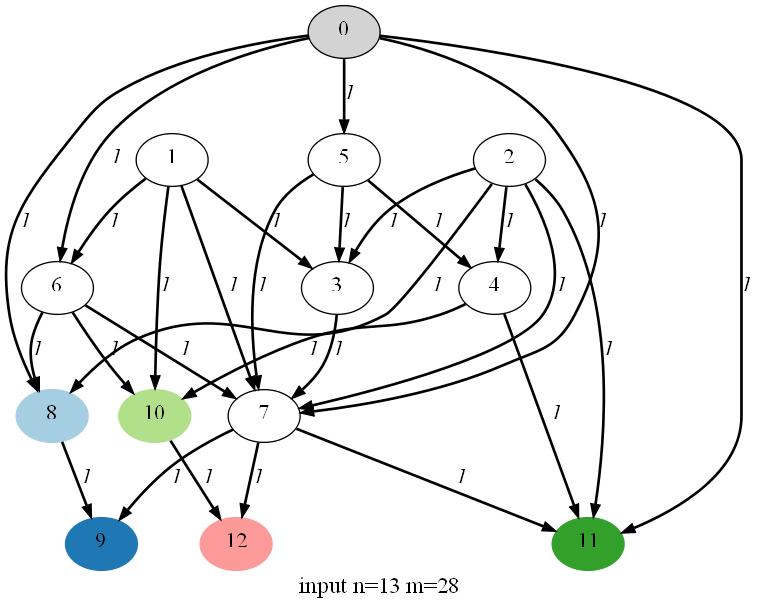
1. No coding operation test case

Figure 24 shows an example which has more than two sink nodes.



1. Three sink nodes test case

As shown in Figure 25, a coding scheme does not utilise all the merged flow as some max-flow to single sink node might exceed the multicast capacity. Flow 0-11 is not used because max-flow from 0 to 11 is 3, which is larger than the multicast capacity 2.



1. Redundancy capacity test case

### Random Test Cases

In order to test the network coding solution, random topologies are generated. The number of node (*n*), edges (*m*), number of sink nodes (*l*) and maximum capacity(*c*) of each edge are specified. There are *m* edges, each edge is started and ended at two different random selected nodes, with a random integer capacity no more than *c*.

As an acyclic graph is required, the Floyd-Warshall algorithm is applied to detect and break the cyclic. Firstly, the standard Floyd-Warshall algorithm is applied to work out the shortest path among all the node pairs. Then it is checked to see if there is a path starting and ending at the same node, if there is, a cycle is detected.

1. **procedure** floyd\_cycle\_detection(graph, n)
2. **begin**
3. **for** k = 1 to n
4. for i = 1 to n
5. for j = 1 to n
6. if graph[i][k] and graph[k][j] then
7. graph[i][j] = True
8. **for** i = 1 to n
9. if graph[i][i] then return “This is a cyclic graph”
10. return “This is a acyclic graph”
11. **end**

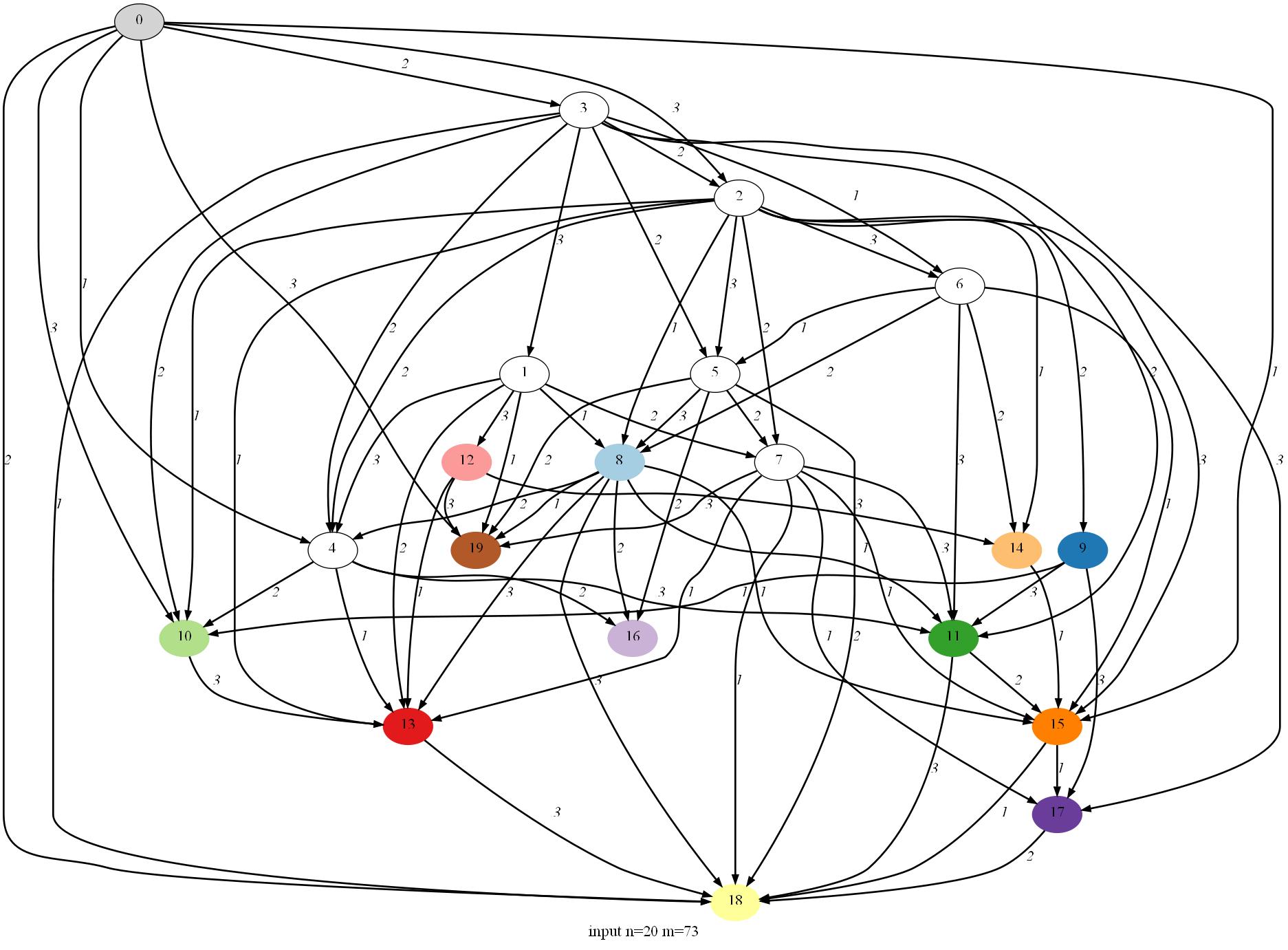
In order to break a cycle, the path reconstruction information is recorded when finding the shortest path. If a cycle is detected, then one edge on the path of the cycle is removed to break that cycle.

1. **procedure** floyd\_break\_cycle (graph, n)
2. begin
3. **for** k = 1 **to** n
4. **for** i = 1 **to** n
5. **for** j = 1 **to** n
6. **if** graph[i][k] **and** graph[k][j] **then**
7. graph[i][j] = True
8. pred[i][j] = k
9. **for** i = 1 to n
10. **if** graph[i][i] **then**
11. remove the edge <pred[i][j], i>
12. end

As there might be more than one cycle covering the same node, and this cycle break method can break only one cycle at a time, therefore it has to iterate until there is no more cycles detected.

In addition, in order to keep the edge distribution balanced, when breaking the cycle, the order of node visitation can be shuffled.

One randomly generated network example is shown in Figure 26, the source node is 0 and the sink nodes are 8 to 19. Each of the sink nodes are filled with different colours. The capacity of each edge is marked at the side of it.



1. Random multicast network

## Validation of the Coding and Routing Scheme

### Capacity Validation

Capacity validation ensures that the number of data blocks at each edge does not exceed the capacity constrain of that edge. This test has been performed in all the experiments. The results show that all the test cases have passed this test.

### Decoding Validation

Decoding validation checks if all the single data blocks could be decoded at every sink node after all the packets arrive at the sink nodes.

Assuming *xor* is used as the coding operation.

Property 1: *a* *xor* *a* = 0

Property 2: *a* *xor* 0 = *a*

*a* *xor* *b xor a*= *a xor a xor b* (*commutative law*) *= 0 xor b* (*Property 1*) *=* b (*Property 2*)

At one sink node, *S*1..*S*n are the data blocks in the data block list of packet *Pi*. Assuming *xor* is used as coding operation, the data block of packet *P* is *S*1 ⊕*S*2 ⊕…⊕*S*n. If Si is the only data block in the data block list of another packet *Pj*, then S*i* could be eliminated from the data block list of *Pi* by operation *Si* ⊕*S*1 ⊕*S*2 ⊕*…*⊕*S*i… ⊕*S*n *= S*1 ⊕*S*2 ⊕*…*⊕*S*i-1⊕⊕*S*i+1… ⊕*S*n.

The decoding is an iteration process. It repeatedly uses a single data block to eliminate itself from the data block list of all the other packets until there is no single data block available. If every data block list contains only one data block then decoding is successful.

The pseudo code for decoding data blocks is listed below:

1. **procedure** decode\_data\_blocks(node v)
2. begin
3. S is the list of all the single data block
4. L is the list of all the data block list of packets at node v
5. **while** S is not empty:
6. *singleton* = pop first data block of S
7. **for** each data block list *sl* in L do
8. **if** *singleton* in *sl*:
9. remove *singleton* from *sl*
10. **if** *sl* has only one data block:
11. S add the only data block in *sl*
12. **if** *w* single data blocks are in *L* **then**
13. Decode successfully
14. **else**
15. Decode failed
16. end

In our experiment, the proposed approach has been applied to 901 valid random networks with 20 nodes. 901 coding and routing schemes have been obtained. Among these schemes, if *xor* is used as coding operation, only 4 coding and routing schemes could not be decoded completely at the sink nodes. In other words, the decoding success rate for the *xor*-only coding scheme is 99.5% in this experiment. For these networks which could not be decoded completely, linear coding operation could be used.

## Experimental Results

### Optimizing Number of Coding Nodes

In most of the random topologies generated, network coding is not necessary to achieve the maximum multicast capacity. Even in the topological network with network coding, the network coding is taking place in only a few links and nodes. The proposed approach can be used to obtain an optimal coding and routing scheme for any network. The strategy in the approach leads to the optimal coding and routing scheme with less number of coding nodes.

#### Comparison with Merging Node Rule

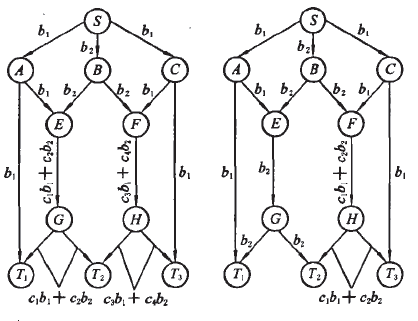
The merging node rule is usually used to identify coding nodes in most of network coding researches. As shown in Table 2, *n* is number of nodes, *m* is number of edge, *l* is number of sink nodes, *w* is number of packets and *t* is number of test cases for the specified *n*, *m*, *l* and *w*. The number of coding nodes identified by our algorithm is much more less than the number of merging nodes.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Random Network | | | | | Coding Node Number of Proposed Methods (avg.) | Merging Node Number (avg.) |
| *n* | *m* | *l* | *w* | *t* |
| 20 | 80 | 12 | 4 | 10 | 0.5 | 5.6 |
| 30 | 90 | 12 | 3 | 5 | 0.2 | 9.0 |
| 40 | 120 | 12 | 3 | 8 | 0.875 | 12.875 |

1. Coding node number of proposed method and merging node number

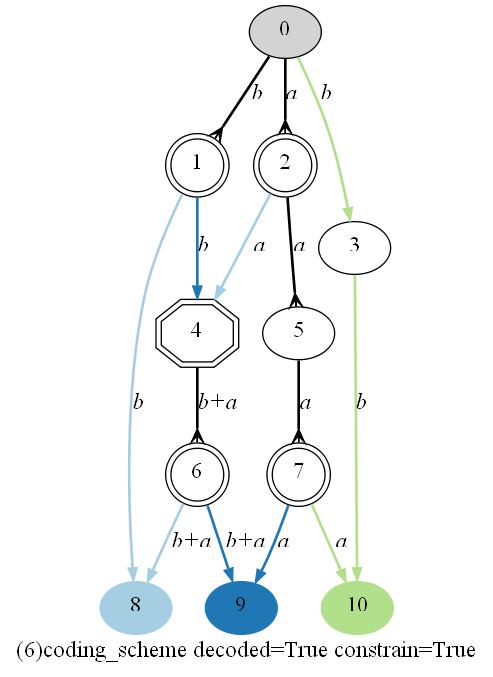
#### Comparison with Key Link Method

Tao proposed a modified max flow algorithm in order to minimise the cost of network coding by reducing the number of key links (overlapped edges). In Figure 27, the coding scheme on the left identifies the coding nodes by the merging node rule. The coding scheme on the right uses their algorithm. The number of coding node is reduced by 50% .



1. Minimal cost network coding example

The coding scheme obtained by the algorithm outlined in this research for the same topologies is shown in Figure 28. There is only one coding node which is same as Tao’s results.



1. Minimal cost coding scheme

### Comparison Number of Coding Links

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Random Network | | | | Real Network | |
| (20,40,12,4) | | (40,120,12,3) | | ISP 1755 (Ebone) (322,1096,4,3) | |
| Best | Avg. | Best | Avg. | Best | Avg. |
| Proposed | 0 | - | 0 | - | 0 | - |
| GA | 0 | 1.20 | 0 | 1.05 | 0 | 0.25 |
| Minimal 1 | 0 | 1.35 | 1 | 1.85 | 0 | 1.05 |
| Minimal 2 | 0 | 1.85 | 0 | 1.90 | 0 | 0.80 |

1. Comparison of coding links required by proposed method and others

In order to compare the results, the networks with same parameters as Kim’s experiments are applied to test the proposed approach . As shown in Table 3, the random network with parameters (20 nodes,40 links,12 sinks,4 packets) and (40 nodes,120 links,12 sinks,3 packets) is chosen in our experiments. And the benchmark data (322 nodes, 1096 links, 4 sinks, 3 packets) is a real network topology of a European back bone (Ebone) from the Rocketfeul project.

The first line shows the results of the our approach. The following results are the Generic Algorithm (GA) by Kim , “Minimal 1” by Fragouli et al and “Minimal 2” by Langberg. . It shows that our algorithm obtained the best number (zero) of coding links for all of the three networks. Furthermore, the proposed approach can obtain the best result in one instance of execution (hence there is no average result in the table). While the best result of the random GA approach is obtained by 20 trails of running .

### Comparison Running Time

With regards execution time, the Generic Algorithm requires at least 15.4 seconds per generation and 1000 generations for a network with 40 nodes . It takes less than one second for all 40 nodes network using the proposed approach in this thesis.

# Conclusion

The fundamental and current research state of Network Coding is examined and the network Max-Flow-Minimum-Cut algorithm is analysed here in this thesis. The research focuses on determining the encoding and decoding nodes in the information transmission and exploring the rules and methods to construct the actual coding and routing scheme. The algorithm to construct the complete coding and routing scheme is outlined. Using the proposed approach, the number of coding nodes and links can be minimized and the running time of the algorithm is extremely short. The dynamic nature of transmitting packets from node to node could be easily applied at routers on existing networks. The experimental results show that based on the proposed algorithm, most of the coding schemes could simply apply the xor as a coding operation.

The summary of the main contributions of thesis are as follows:

1. Increased the assumption capacities of links from unit to integer, a set of new definitions and concepts are introduced for network coding including merged max-flows, overlapped edge, target set of packet, target set of edge, forward operation, multicast operation and coding operation.
2. Brought the new idea of building a multicast scheme by following the max-flow paths to every single sink node. The original network is simplified to merged max-flow graph. An efficient algorithm is proposed to construct a network coding and routing scheme on the simplified network.
3. Introduced the strategy of best-matching target sets between the packet and edge. This mechanism enables the algorithm to maximise the efficiency of every edge and reduce the overall network resource usage.
4. Introduced the forwarding-multicasting-coding mechanism to minimise the number of coding operation. Consequently, the algorithm identifies coding nodes dynamically during the process of constructing the coding and routing scheme.

# References

Ahlswede, R., Cai, N., Li, S. & Yeung, R., 2000. Network Information Flow. *IEEE Transactions on Information Theory*, 46(4), pp.1204-16.

Chow, P.A., Wu, Y. & Jain, K., 2003. Practical network coding. In *Proceedings of 41st Annual Allerton Conference on Communication, Control, and Computing*. Monticello, IL, 2003.

Corman, T.H., Leiserson, C.E. & Rivest, R.L., 1991. *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press.

Edmonds, J., 1973. Edge-Disjoint Branchings. *Combinatorial Algorithms*.

Elias, P., Feinstein, A. & Shannon, C.E., 1956. A note on the maximum flow through a network. *IEEE Transactions on Information Theory*, pp.117-19.

Fragouli, C. & Soljanin, E., 2007. Network Coding Fundamentals. *Foundations and Trends in Networking*, 2(1), pp.1 - 133.

Fragouli, C., Soljanin, E. & Shokrollahi, A., 2004. Network coding as a coloring problem. In *Proceedings of 38th Annual Conference on Information Sciences*. Princeton, NJ, 2004.

Gkantsidis, C. & Rodriguez, P.R., 2004. *Network coding for large scale content distribution*. Microsoft Research.

Ho, T., Karger, D. & Medard, M., 2003. The benefits of coding over routing in a randomized setting. In *IEEE International Symposium on Information Theory*., 2003.

Ho, T., Leong, B. & Koetter, R., 2004. Byzantine modification detection in multicast networks using randomized network coding. In *IEEE International Symposium on Information Theory (ISIT 2004)*. Chicago, 2004.

Ho, T., Muriel, M. & Ralf, K., 2005. An information theoretic view of network management. *IEEE Transactions on Information Theory*, 51(4), pp.1295-312.

Jaggi S P, C.A.J.K., 2003. Low complexity algebraic network multicast codes. In *ISIT 2003*. Yokohama, 2003.

Jain, K., Mahdian, M. & Salavatipour, M., 2003. Packing Steiner trees. In *The 14th ACM-SIAM symposium on Discrete Algorithms (SODA) 2003*. Baltimore, MD, 2003.

Katti, S. et al., 2005. The importance of being opportunistic: Practical network coding for wireless environments. Allerton, 2005.

Kim, M., Ahn, C.W., Médard, M. & Effros, M., 2006. On Minimizing Network Coding Resources: An evolutionary approach. In *Network Coding Workshop*., 2006.

Kim, M. et al., 2007. Evolutionary approaches to minimizing network coding resources. In *Proc. IEEE Infocom*., 2007.

Koetter, R. & Medard, M., 2002. Beyond routing: an algebraic approach to network coding. In *INFOCOM 2002*. New York, 2002.

Koetter, R. & Medard, M., 2003. An algebraic approach to network coding. *IEEE ACM Trans on networking*, 11(5).

Langberg, M., Sprintson, A. & Bruck, J., 2005. The encoding complexity of network coding. In *IEEE International Symposium on Information Theory (ISIT)*. Adelaide, Australia, 2005.

Li, S.Y.R., Yeung, R.W. & Cai, N., 2003. Linear network coding. *IEEE Trans on Information Theory*, 49(2).

Lun, D., Ratnakar, N. & Koetter, R., 2005. Achieving minimum-cost multicast: A decentralized approach based on network coding. In *Proc of the IEEE INFOCOM 2005*. Miami, 2005. IEEEComputer Society.

Lun, D., Ratnakar, N. & Koetter, R., 2006. Minimum-Cost multicast over coded packet networks. *IEEE Trans. On Information Theory*, 52(6), pp.2608-23.

Menger, K., 1927. Zur allgemeinen Kurventheorie. *Fundamenta Math*, 10, pp.96-115.

Pakzad, P., Fragouli, C. & Shokrollahi, A., 2005. Coding Schemes for Line Networks. In *IEEE International Symposium on Information Theory (ISIT'05)*. Adelaide, Australia, 2005.

Raymond, W.Y., 2008. Network Coding for Error Correction and Security. In *NetCod 2008*. Hong Kong, 2008.

Sanders, P., Egner, S. & Tolhuizen, L., 2003. Polynomial Time Algorithms for Network Information Flow. In *Symp. Parallelism in Algorithms and Architectures (SPAA)*. San Diego, 2003. ACM.

Spring, N., Mahajan, R. & Wetherall, D., 2002. Measuring ISP topologies with rocketfuel. In *Proc. ACM/SIGCOMM*., 2002.

Sun, Y., 2005. *An Overview of Network Information Flow Problem*. Available at: http://www.maths.qmul.ac.uk/~pjc/csgnotes/network.pdf

Tao, S., Huang, J., Yang, Z. & Cheng, W., 2008. An improved algorithm for minimal cost network coding. *Journal of Huazhong University of Science and Technology (Natural Science Edition)*, 36(5), pp.1-5.

Tao, S. et al., 2008. Maxflow-based Routing Algorithm for Network Coding Multicast. *Computer Science*, 35(6).

Wang, J., Liu, J. & Wang, X., 2008. Performance Analysis of Multicast Routing Algorithm Based on Network Coding. *Journal of Electronics & Information Technology*, 30(11).

Wu, Y., Chou, P. & Jain, K., 2004. A comparison of network coding and tree packing. *IEEE International Symposium on Information Theory*, p.143.

Wu, Y., Chou, P. & Kung, S., 2005. Information Exchange in Wireless Networks with Network Coding and Physical-layer Broadcast. In *Proc. 39th Annual Conference on Information Sciences and Systems*., 2005.

Yuan, J., Li, Z. & Yu, W., 2005. A cross-layer optimization framework for multicast in multi-hop wireless networks. In *Proc. of First International Conference of Wireless Internet*. Budapest, Hungary, 2005.

Zosin, L. & Khuller, S., 2002. On directed Steiner Trees. In *13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*., 2002.