NETWORK CODING IN DISTRIBUTED, DYNAMIC, AND WIRELESS ENVIRONMENTS: ALGORITHMS AND APPLICATIONS

A Dissertation

by

MOHAMMAD ASAD REHMAN CHAUDHRY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2011

Major Subject: Electrical Engineering

Network Coding in Distributed, Dynamic, and Wireless Environments: Algorithms and Applications

NETWORK CODING IN DISTRIBUTED, DYNAMIC, AND WIRELESS ENVIRONMENTS: ALGORITHMS AND APPLICATIONS

A Dissertation

by

MOHAMMAD ASAD REHMAN CHAUDHRY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Alexander Sprintson
Committee Members,	A. L. Narasimha Reddy
	Jennifer Lundelius Welch
	Deepa Kundur
Head of Department,	Costas N. Georghiades

December 2011

Major Subject: Electrical Engineering

ABSTRACT

Network Coding in Distributed, Dynamic, and Wireless Environments:
Algorithms and Applications. (December 2011)
Mohammad Asad Rehman Chaudhry, B.Sc., The University of Engineering and Technology;
M.Sc., The University of Engineering and Technology
Chair of Advisory Committee: Dr. Alexander Sprintson

The network coding is a new paradigm that has been shown to improve throughput, fault tolerance, and other quality of service parameters in communication networks. The basic idea of the network coding techniques is to relish the "mixing" nature of the information flows, i.e., many algebraic operations (e.g., addition, subtraction etc) can be performed over the data packets. Whereas traditionally information flows are treated as physical commodities (e.g., cars) over which algebraic operations can not be performed. In this dissertation we answer some of the important open questions related to the network coding. Our work can be divided into four major parts.

Firstly, we focus on network code design for the dynamic networks, i.e., the networks with frequently changing topologies and frequently changing sets of users. Examples of such dynamic networks are content-distribution networks, peer-to-peer networks, and mobile wireless networks. A change in the network might result in infeasibility of the previously assigned feasible network code, i.e., all the users might not be able to receive their demands. The central problem in the design of a feasible network code is to assign local encoding coefficients for each pair of links in a way that allows every user to decode the required packets. We analyze the problem of maintaining the feasibility of a network code, and provide bounds on the number

of modifications required under dynamic settings. We also present distributed algorithms for the network code design, and propose a new *path-based assignment* of encoding coefficients to construct a feasible network code.

Secondly, we investigate the network coding problems in wireless networks. It has been shown that network coding techniques can significantly increase the overall throughput of wireless networks by taking advantage of their broadcast nature. In wireless networks each packet transmitted by a device is broadcasted within a certain area and can be overheard by the neighboring devices. When a device needs to transmit packets, it employs the *Index Coding* that uses the knowledge of what the device's neighbors have heard in order to reduce the number of transmissions. With the Index Coding, each transmitted packet can be a linear combination of the original packets. The Index Coding problem has been proven to be NP-hard, and NP-hard to approximate. We propose an efficient exact, and several heuristic solutions for the Index Coding problem. Noting that the Index Coding problem is NP-hard to approximate, we look at it from a novel perspective and define the *Complementary Index Coding* problem, where the objective is to maximize the number of transmissions that are saved by employing coding compared to the solution that does not involve coding. We prove that the Complementary Index Coding problem can be approximated in several cases of practical importance. We investigate both the *multiple unicast* and *multiple multicast* scenarios for the Complementary Index Coding problem for computational complexity, and provide polynomial time approximation algorithms.

Thirdly, we consider the problem of accessing large data files stored at multiple locations across a content distribution, peer-to-peer, or massive storage network. Parts of the data can be stored in either original form, or encoded form at multiple network locations. Clients access the parts of the data through simultaneous downloads from several servers across the network. For each link used client has to pay

some cost. A client might not be able to access a subset of servers simultaneously due to network restrictions e.g., congestion etc. Furthermore, a subset of the servers might contain correlated data, and accessing such a subset might not increase amount of information at the client. We present a novel efficient polynomial-time solution for this problem that leverages the matroid theory.

Fourthly, we explore applications of the network coding for congestion mitigation and overflow avoidance in the global routing stage of Very Large Scale Integration (VLSI) physical design. Smaller and smarter devices have resulted in a significant increase in the density of on-chip components, which has given rise to congestion and overflow as critical issues in on-chip networks. We present novel techniques and algorithms for reducing congestion and minimizing overflows.

قُلْ إِنَّ صَلَاتِي وَنُسُكِي وَمَحْيَايَ وَمَمَاتِي لِلَّهِ رَبِّ الْعَالَمِينَ ﴿١٦٢﴾

لَا شَرِيكَ لَهُ ۖ وَبِذَٰلِكَ أُمِرْتُ وَأَنَا أَوَّلُ الْمُسْلِمِينَ ﴿١٦٣﴾

Dedicated to my parents, and my wife

ACKNOWLEDGMENTS

I would like to acknowledge the cooperation and support provided to me by my advisor Dr. Alex Sprintson. I learnt a lot from him, not only for my research, but for other parts of my life as well. Long discussions and debates with him always resulted in something very fruitful. He has been an excellent mentor teaching the skills to lead a successful life. He has always been deeply involved in my journey to explore new depths of knowledge and learning. I would also like to thank the other members of my advisory committee, Dr. Jennifer Welch, Dr. Deepa Kundur, and Dr. A.L. Narasimha Reddy, for their feedback and support. Dr. Welch was always there whenever I needed her guidance and help. Dr. Kundur has always been a very positive and supportive influence throughout my PhD. Dr. Reddy always provided very useful feedback. I would also like to thank my friend and teacher Dr. Don Halverson for intriguing discussions about the wonderful world of mathematics. I am also thankful to many others whose names are not explicitly mentioned. The material is based upon research partly supported by the National Science Foundation under Grant No. 0954153.

I do not have words to express my gratitude toward my most respected parents. It is solely because of them that I am able to go so far in my career. They have stood behind me throughout my life and they were there when I needed them the most. I pray that their blessing may continue to sail with me for the rest of my life. All this work was impossible without the relentless support and prayers of my dearest sisters throughout my career. Finally, whatever I say shall be inadequate to reveal my gratefulness to my beloved wife for all she did during the course of our life.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

A. Network Coding

The network coding has gained a significant interest from the research community since the first paper by Alshwede et al. in 2000 [1], as it has been shown to improve throughput, resilience and fault tolerance, and other quality of service parameters in communication networks as compared to the traditional techniques. The basic idea of the network coding is to allow algebraic operations on the data packets at the network nodes (routers, relays etc), as compared to the traditional schemes in which intermediate nodes can only forward packets. In traditional approaches data packets are treated as physical commodities and data flow is dealt in the same way as commodity flow, e.g., cars on a highway over which algebraic operations (addition, subtraction etc) can not be performed. Whereas the network coding takes advantage of the fact that data flow is different from the commodity flow and the data packets, in contrast to the physical commodities, can be subject to algebraic operations (addition, subtraction etc). In the network coding the packet transmitted by a network node is a function of the packets received at its incoming links as shown in Figure 1. Consider a butterfly network where each link can just send one packet per time unit as shown in Figure 2. This network was presented by Alshwede et al. in [1]. Note, that in this butterfly network there are two edge-disjoint paths from source to each of the terminals. Namely, the two edge-disjoint paths from the source s to the terminal t_1 are $s-w-t_1$ and $s-x-y-z-t_1$, and the two edge-disjoint paths from the source s to the terminal t_2 are $s-x-t_2$ and $s-w-y-z-t_2$. It is obvious that the source

---

The journal model is *IEEE Transactions on Information Theory.*

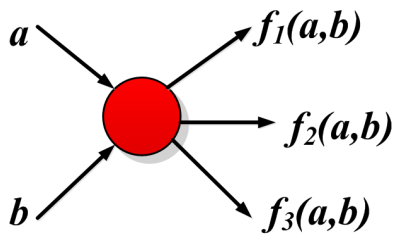

Fig. 1. The network coding.

s can send two packets per time unit to either terminal t_1 or terminal t_2 by sending one packet per time unit on each of the corresponding edge-disjoint paths, but note that using traditional routing approach the source s can not send two packets per time unit to both of the terminals t_1 and t_2 simultaneously due the the bottleneck link $y - z$. In case the source can subdivide a packet into smaller subpackets (this technique is referred to as *fractional routing*) then the rate at which source can send packets to both of the terminals is bounded by 1.5 packets per time unit. By using the network coding, i.e., combining packets at the network node y (e.g., addition over a finite field), the source can send two packets per time unit to both of the clients. Terminal t_1 shall be receiving two packets a and $a + b$ per time unit, i.e., it receives packet a directly and can decode packet b by a simple subtraction $(a + b) - a$ of the two received packets. Similarly terminal t_2 can get both the packets a and b from its received packets b and $a + b$. In short an information flow rate of two packets per time unit can not be achieved with traditional routing, whereas the network coding achieves this rate.

Another application of the network coding is wireless networks. For example consider a wireless network consisting of two nodes and a relay as shown in Figure 3.

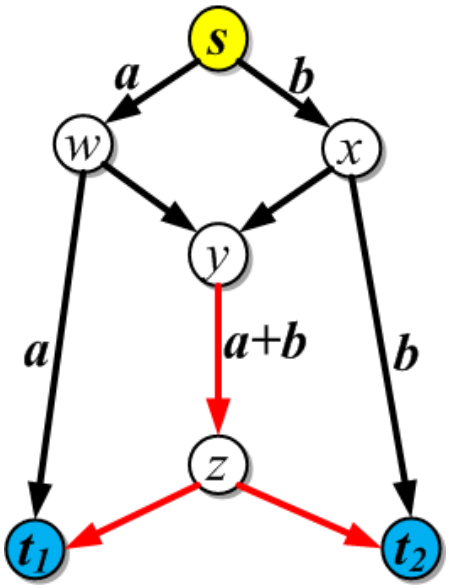

Fig. 2. The butterfly network [1].

The nodes can not communicate directly, and need the help of the relay to exchange data. Traditional approach requires four transmissions in the system. Two transmissions to deliver packet a from node 1 to node 2, as firstly node 1 transmits packet a to the relay, and then the relay transmits this packet to the node 2. Similarly two transmission are required to deliver packet b from node 2 to node 1. By using the network coding technique, the number of transmissions in the network can be reduced from four to three. In case of the network coding approach the relay firstly receives packet a from node 1 and waits until it receives packet b from node 2, and after receiving both the packets it creates a combined packet $a + b$ and broadcasts it to both the nodes. Note, it results in total of three transmissions in the network one transmission from each of the nodes, and one transmission from the relay. Each node after receiving packet $a + b$ can decode either packet a or b based on whatever it already has. For example node 1 already has packet a and it can use $(a + b) - a$ to

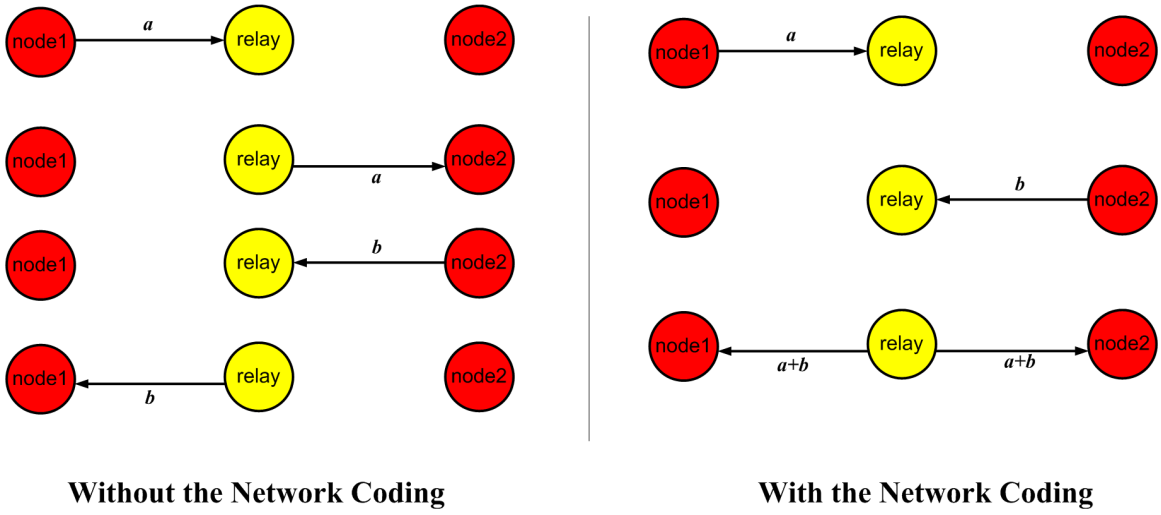

Fig. 3. The network coding in wireless setup.

decode packet b, and similarly node 2 can decode packet a.

The reduction in the number of transmissions in the network can result in reducing overall energy consumption, and interference in the network. Furthermore, the network coding in this scenario results in efficient utilization of bandwidth.

Distributed data storage is crucial for providing secure and reliable data access through redundancy. It has been shown that the network coding based regenerating codes can significantly reduce the bandwidth required for repairing the failed disks in a distributed storage system [2].

Another interesting application of the network coding is in Very Large Scale Integration (VLSI) circuit design where congestion and overflows are one of the major challenges due to increase in density of on-chip components. The increase in density of on-chip components results from decreasing chip size and increasing demand of chip functionalities. The network coding can be utilized in the global routing stage of VLSI physical design to encourage sharing of circuits or wires among different

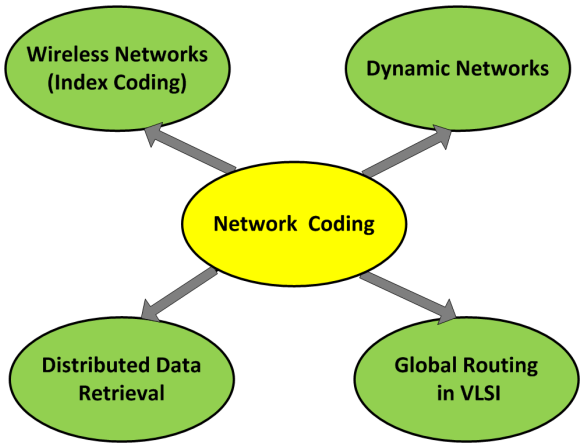

Fig. 4. Classification of the contributions presented in the dissertation.

circuits [3].

B. Contributions

This dissertation focuses on four different aspects of the network coding as shown in Figure 4. Contribution of the dissertation can be classified into following four categories:

1. Efficient Network Coding Algorithms For Dynamic Networks

In this part of the dissertation we present design of efficient multicast network codes for dynamic networks. Examples of such dynamic networks are content-distribution networks, peer-to-peer networks, and mobile wireless networks. Dynamic nature of networks can arise because of many scenarios like varying capacity of the links due to congestion, hardware failures resulting in loss of communication links, addition of new communication links to compensate for broken links, and varying number of end users connected to a network. To the best of our knowledge, the algorithms

presented are the first distributed and fully deterministic algorithms for the network code assignment that can deal with the dynamic nature of the networks.

One of the central problems in the network coding is to design an efficient algorithm that assigns the local encoding coefficients in a way that allows each user to decode the required packets. Such a network code is called a *feasible network code*. We focus on maintaining the feasibility of a given network code upon an addition of a new user, or a change in the network topology. Any change in the network might result in infeasibility of the previously assigned feasible network code, i.e., all the users might not be able to receive their demands. Our goal is to minimize the number of encoding coefficients that are required to be modified (after a change in the network) to keep the network code feasible. A smaller number of required changes in the encoding coefficients will allow the coding network to adjust for a new user or a change in the network topology more efficiently, and also reduce the disruption to existing users.

Formally, we consider the problem of minimizing the number of encoding coefficients that are required to be changed to accommodate a new user or respond to a change in the underlying network topology. We focus on three natural questions. One, what is the computational complexity of this minimization problem. Two, if it is possible to establish tight upper and lower bounds on the minimum number of changes required in encoding coefficients to cope up with dynamic nature of the network. Three, if it is possible to design an algorithm that efficiently handle frequent network changes. Our work deals with these questions in that order.

In order to understand how changes in a network can affect the feasibility of a network code consider the following examples. Consider a mulitcast network with one source and four terminals $t_1,\ t_2,\ ,t_3,\ t_4$ as shown in Figure 5(a). The source wants to send two packets a and b to all the terminals. When a new terminal $\hat{t}$ joins

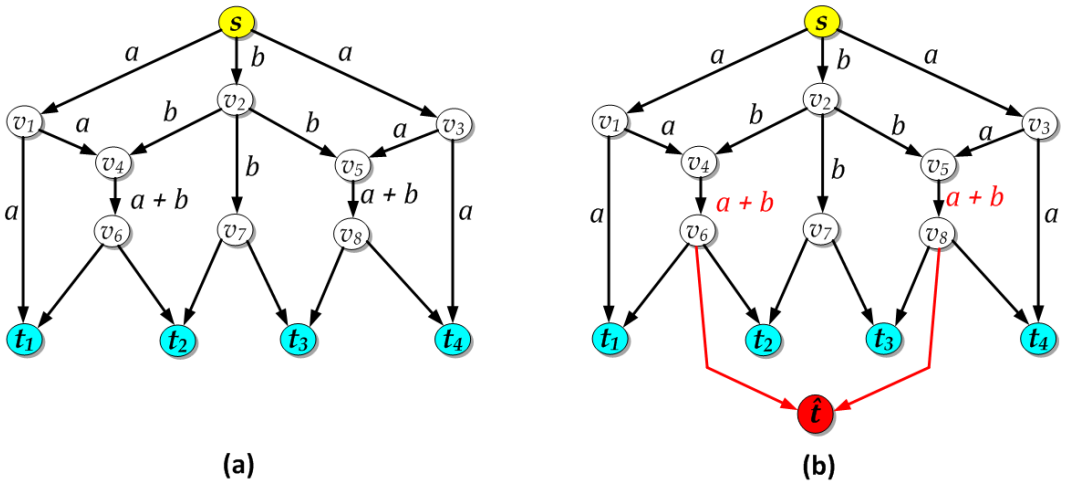

Fig. 5. (a) A network and corresponding feasible network code. (b) A new network $\hat{\mathbb{N}}$ after a new node $\hat{t}$ joins.

the network and connects to nodes v_6 and v_8, the previously feasible network code, as shown in Figure 5(a), becomes infeasible as the new terminal is unable to decode the original packets sent by the source. Note that the new terminal $\hat{t}$ is receiving the packet $a + b$ from both of its incoming links, hence it can neither decode packet a nor packet b. In case of the failure of an edge (v_2, v_7) for the network shown in 6(a) the effected terminals t_2 and t_3 respond to the loss of a link by connecting to nodes v_6 and v_8 respectively in order to maintain the minimum connectivity requirement to be able to receive two packets from the source. This change in topology makes the network code infeasible as shown in 6(b). Accordingly, there is need to modify a number of network coding coefficients to make the network code feasible again. Keeping the modifications to a minimum is practically important as it will allow the coding network to adjust for a new user or a change in the network topology more efficiently and also reduce the disruption for existing users.

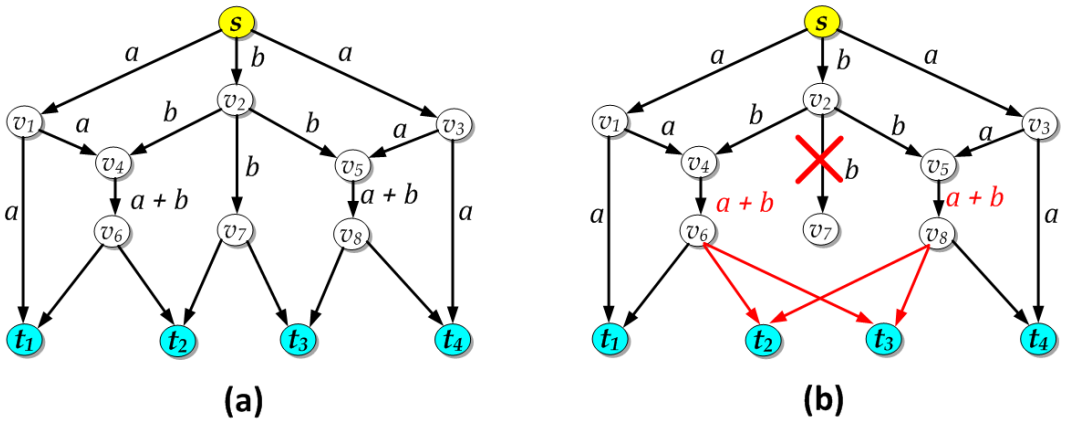

Fig. 6. (a) A network $\mathbb{N}$ and a feasible network code $\mathbb{C}$ for $\mathbb{N}$. (b) A new network $\hat{\mathbb{N}}$ after failure of an edge.

We first analyze the problem of maintaining the feasibility of a network code by minimizing the encoding coefficients that need to be modified (after a change in the network) to keep the network code feasible. Second, we present lower and upper bounds on the number of modifications required in case of an addition of a new user or a failure of an edge. Third, we analyze the computational complexity of the problem in hand, and prove it to be NP-complete. Fourth, we present an algorithm for the network code design, and propose a new *path-based assignment* of encoding coefficients to construct a feasible network code. Fifth, we present a new method for assignment of encoding coefficients which is based on the prime numbers. The presented assignment scheme is distributed in nature, and does not require full knowledge of the network topology. Sixth, we present extensive simulation studies on practical networks to show the advantage of the proposed schemes in practical scenarios.

2. Index Coding

Another aspect of our work is investigation of the network coding in single hop wireless networks. The related problem, referred to as the *Index Coding problem* [4, 5], has recently attracted a significant interest from the research community [6]. It has been shown that the Index Coding techniques can significantly increase the overall throughput of wireless networks by taking advantage of their broadcast nature. In wireless networks each packet transmitted by a device is broadcasted within a certain area and can be overheard by the neighboring devices. When a device needs to transmit packets, it employs the *opportunistic coding* approach that uses the knowledge of what the device's neighbors have heard in order to reduce the number of transmissions. With opportunistic coding approach, each transmitted packet can be a linear combination of the original packets over a certain finite field. An instance of the Index Coding problem comprises of a server, and a set of clients. The server holds a set of n packets $P = \{p_1, \ldots, p_n\}$. Each client is interested in a certain subset of packets available at the server, and might have a (different) subset of packets available as side information. The server can broadcast the packets in P or encoding thereof. The goal is to find a scheme that requires the minimum number of transmissions to satisfy the requests of all clients. The server can broadcast the packets in P or encoding thereof. The objective is to identify a scheme that satisfies the demands of all clients with the minimum possible number of transmissions. Each client is represented by a pair *(Want, Has)*, where *Want* is the set of packets required by the client, and *Has* is the *side information* or the set of packets available to the client.

Figure 7 shows an instance of the Index Coding problem. The central node, referred to as a server, needs to deliver five packets $p_1, \ldots, p_5$ to five clients $c_1, \ldots, c_5$; packet p_i needs to be delivered to client c_i. Each client c_i has access to some *side*

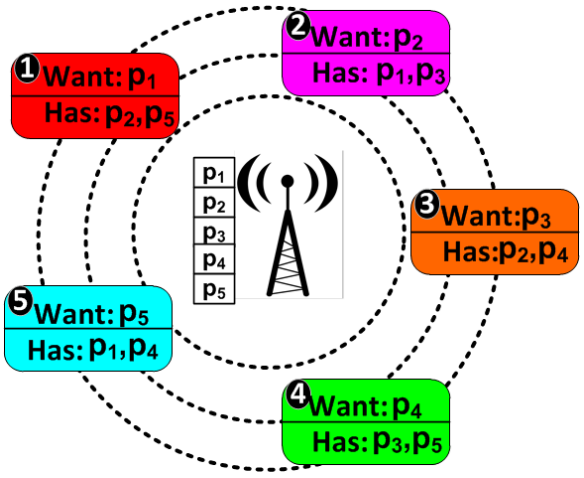

Fig. 7. An instance of the Index Coding problem including a server and five clients. Server can transmit the packets $p_1, \cdots, p_5$ or their linear combinations to satisfy the demand of the clients.

information which might be the packets overheard (cached) from prior transmissions. These packets are included in the client's "Has" set. It is easy to verify that all clients can be satisfied by broadcasting three packets $p_1 + p_2$, $p_3 + p_4$, and p_5 (all additions are over $GF(2)$). Whereas in traditional approach all the five packets $p_1, \ldots, p_5$ are needed to be transmitted, hence the Index Coding technique reduces the number of transmissions from 5 to 3.

The Index Coding problem has been proven to be *NP-hard*, and *NP-hard to approximate*. First, we propose an efficient exact, and several heuristic solutions for the Index Coding problem. Our numerical study suggests that the exact solutions can be efficiently identified for small instances, while the heuristic solutions with small computation time can achieve near optimal performance for larger instances. We then focus on finding approximate polynomial time solutions for the Index Coding problem with mathematically proven guarantees.

Noting that the Index Coding problem has been proven to be NP-hard to approximate, we explore at it from a novel perspective and define the *Complementary Index Coding* problem. The goal of the Complementary Index Coding problem is to maximize the number of *saved transmissions*, i.e., the number of transmissions that are saved by employing encoding compared to the solution that does not involve coding. For the instance of the Index Coding problem shown in Figure 7, the number of saved transmissions is 2. Second, we prove that the Complementary Index Coding problem can be approximated in several cases of practical importance. We investigate both the *multiple unicast* and *multiple multicast* scenarios for the Complementary Index Coding problem. In the multiple unicast scenario, each packet is requested by a single client; while in the multiple multicast scenario, each packet can be requested by several clients. Third, we present polynomial time approximation algorithms for finding *scalar* and *fractional* linear solutions for the multiple unicast scenario. Fourth, we show that for the multiple multicast scenario finding an approximation solution is NP-hard, and the multiple multicast scenario is NP-hard to approximate as well.

Fifth, we focus on finding *sparse* solutions to the Index Coding problem with mathematically provable guarantees. In a sparse solution each transmitted packet is a linear combination of at most two original packets. With the sparse Index Coding, the encoders and decoders can be implemented very efficiently which makes it attractive for practical applications. The sparse Index Coding can be implemented over a small field ($GF(2)$), which results in significant reduction in the size of the packet headers and the associated overhead. Consider another instance of index coding problem shown in Figure 8. An optimal solution for satisfying three clients c_1, c_2, c_3, is to broadcast one packet $p_1 + p_2 + p_3$. But under the restriction of encoding at most two packets, the server can satisfy three clients in two transmissions: $p_1 + p_2$, and p_3, i.e., still a transmission is saved compared to the traditional schemes. We analyze

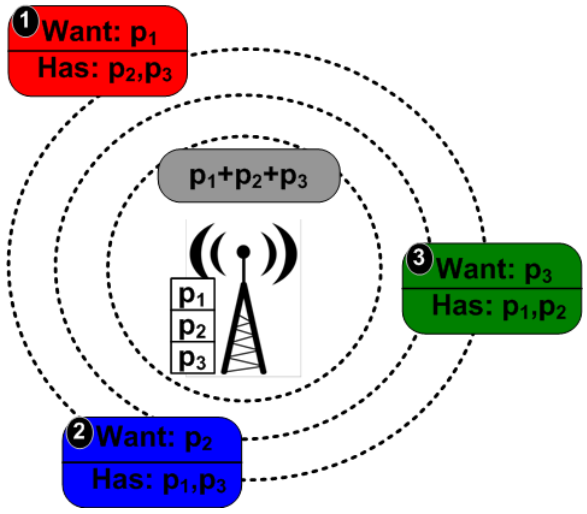

Fig. 8. An instance of the Index Coding problem. The optimum non-sparse solution is to broadcast one packet $p_1 + p_2 + p_3$. However an optimal solution to sparse Index Coding problem requires two transmissions, i.e., $p_1 + p_2$, and p_3.

both *scalar* and *fractional* versions of the problem, and provide polynomial time solution. For the scalar case, we present a polynomial time algorithm that achieves an approximation ratio of $2 - \frac{1}{\sqrt{n}}$. For the fractional case, we present a polynomial time algorithm that provides the optimal solution to the problem.

Sixth, we perform an extensive experimental study which demonstrates that our algorithms achieve good performance in practical scenarios.

3. Distributed Data Retrieval

In many practical settings, clients need to access large data files stored at multiple locations across the network. For example, in content distribution networks the data is stored across multiple geographical locations to enable efficient access by multiple clients. Similarly, in peer-to-peer networks clients retrieve popular files such as movies from their peers. In mass storage systems, the data is distributed throughout the

network to increase the reliability and resilience to failures. When a client needs to obtain a copy of a large data object, it initiates simultaneous downloads from multiple servers.

In this part of the dissertation, we consider the problem of accessing large data objects (e.g., multimedia files, datasets, etc) stored at multiple network locations. Each data object can be divided into a number of fixed-size blocks, which are stored at servers across the network.

There are three major approaches for distributing the data across the servers. The first approach uses data replication or *mirroring*. With this approach, several copies of each block are stored on different servers across the network. A client needs to identify a subset of the nearby servers that collectively store all the required blocks and obtain one copy of each block through simultaneous downloads.

The second approach uses erasure correcting codes to generate parity check blocks. With this approach, k original blocks are encoded into n blocks using a *Maximum Distance Separable (MDS)* code, such that any k out of n blocks are sufficient for decoding the content of the file. A client needs to locate k nearby servers, and initiate simultaneous downloads to obtain k different coded blocks. The blocks are then decoded to obtain the content of the required file.

The third approach is to use a *general linear coding* scheme, which is not necessarily an MDS coding scheme. Such schemes are used, for example, in distributed storage systems [2]. In such schemes, a client needs to identify a subset of servers that collectively store enough data to be able to obtain the content of the original file. More specifically, suppose that the original file is divided into k blocks and that each block stored at a server is a linear combination of k original blocks. Thus, a client needs to simultaneously download data from k servers that store k linearly independent combinations of the original blocks. The contents of the original file can

then be decoded by performing linear operations on the obtained data. Note that the general linear coding scheme includes the first two approaches as special cases.

Clients access the data through simultaneous downloads from several servers across the network. For each link used client has to pay some cost. The cost can capture price of usage, link congestion etc. A client might not be able to access a subset of servers simultaneously due to network restrictions e.g., congestion, capacity overload etc. Furthermore, a subset of the servers might contain correlated data, and accessing such a subset might not increase amount of information at the client.

We focus on the general linear coding settings and consider the problem of minimizing the total cost of downloading the contents of a file from multiple servers. We assume that each link in the network is associated with a certain cost and has capacity constraints. Our goal is to find a set of k paths of minimum total cost that connect a subset of data servers with the client. The k paths should satisfy the following constraints:

1. Each path connects a data server and the client;
2. Each path is used for downloading a single data block;
3. The k downloaded data blocks are linearly independent;
4. The number of paths that share a single link cannot exceed the capacity of that link.

Note that in order to solve this problem we need to select a subset of data servers and the corresponding paths to the client through which the data will be downloaded. We refer to this problem as the *Distributed Data Retrieval* (DDR) problem.

Figure 9 demonstrates three approaches for storing three original blocks, a, b, and c across four servers and the corresponding instances of Problem DDR. Figure 9(a)

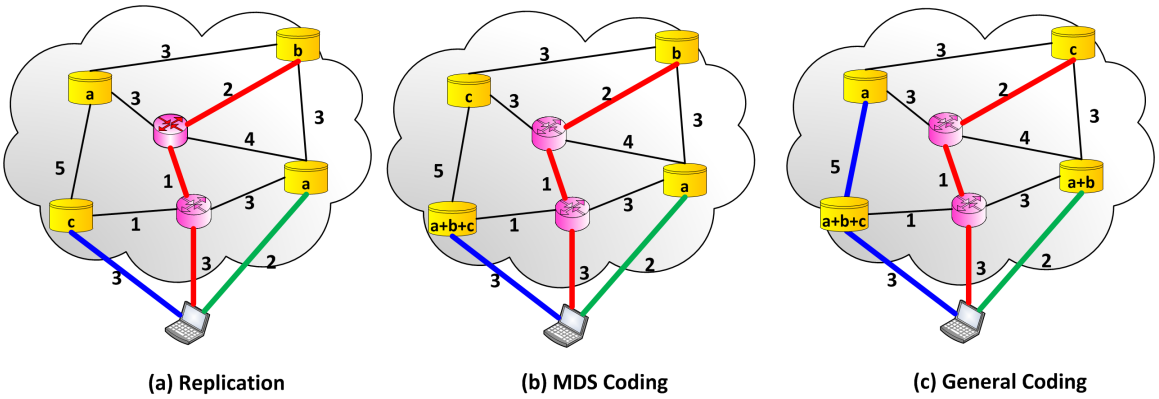

Fig. 9. Storage schemes considered. (a) Replication-based approach. (b) An approach based on MDS codes. (c) An approach based on general linear codes. For each scenario, the optimal set of paths to retrieve packets a, b, and c is shown by thick lines.

shows a replication based approach. With this approach a client needs to find three disjoint paths, one originating at a server that stores block a, second at a server that stores block b, and third at a server that stores block c. Figure 9(b) demonstrates the approach where the data is stored using an *MDS* code (here, all operations are performed over $GF(2)$). With this approach, the client needs to find three disjoint paths of the minimum total cost to any three distinct servers. The general coding approach is depicted in Figure 9(c). In this scheme, the paths must originate at servers that store linearly independent combinations. In all figures, disjoint paths of minimum total cost are shown by thick lines.

We present a novel efficient polynomial-time solution for the *Distributed Data Retrieval* (DDR) problem that leverages the matroid theory. Our experimental study shows the advantage of our solution over alternative approaches.

4. Efficient Rerouting Algorithms for Congestion Mitigation

In this part of the dissertation we investigate the applications of the network coding for congestion mitigation and overflow avoidance in the global routing stage of Very Large Scale Integration (VLSI) physical design. With the advent of smaller devices, a significant increase in the density of on-chip components has raised congestion and overflow as critical issues in VLSI physical design automation. We present novel techniques for reducing congestion and minimizing overflows. Our methods are based on ripping-up nets that go through the congested areas and replacing them with congestion-aware topologies. Our contributions can be summarized as follows. First, we present several efficient algorithms for finding *congestion-aware* Steiner trees, i.e., trees that avoid congested areas of the chip. Second, we show that the novel technique of network coding can lead to further improvements in routability, reduction of congestion, and overflow avoidance. Thirdly, we present an algorithm for identifying efficient *congestion-aware network coding topologies*. We evaluate the performance of the proposed algorithms through extensive simulations using the International Symposium on Physical Design (ISPD) routing benchmarks.

To understand how network coding can help in congestion avoidance in VLSI design, consider a routing instance depicted in Figure 10(a). In this example, we need to route two nets, one net connecting source s_1 with terminals t_1, t_2, and t_3, and the other net connecting source s_2 with the same set of terminals. The underlying routing graph is represented by a grid $G(V, E)$ as shown in Figure 10(a). Suppose that due to congestion each edge of this graph has a residual capacity of one unit, i.e., each edge can accommodate only a single wire. It is easy to verify that only one net can be routed without an overflow. For example, Figure 10(b) shows a possible routing of a net that connects s_1 with terminals t_1, t_2, t_3. Figure 10(c) shows

that routing of both nets results in an overflow. In this example, two nets transmit different signals, a and b, over separate Steiner trees. Figure 10(d) shows that the network coding approach allows to route both nets without violating edge capacities. With this approach, the terminal t_1 creates a new signal, $a \oplus b$, which is delivered to terminals t_2 and t_3, while the signals a and b are delivered to terminals t_2, and t_3 directly. It is easy to verify that with this scheme each terminal can decode the two original signals, a and b.

The example above indicates that network coding can offer two distinct advantages. First, it has a potential of solving very difficult cases. In some instances, the network coding technique allows to solve cases that cannot be routed by conventional techniques. Second, it can achieve a decrease in the total wirelength, and, at the same time, alleviate the routing congestion.

C. Dissertation Outline

In Chapter II, we discuss the network coding for dynamic networks. In Chapter III we introduce the Index Coding problem and present the corresponding model. Chapter IV describes a set of efficient algorithms for the Index Coding problem . In Chapter V we discuss the Complementary Index Coding problem. In Chapter VI, we discuss the Sparse Index Coding problem. Chapter VII deals with the distributed data retrieval problem. Chapter VIII discusses applications of the network coding in congestion mitigation and overflow avoidance in the VLSI circuit design. We present conclusions in Chapter IX.

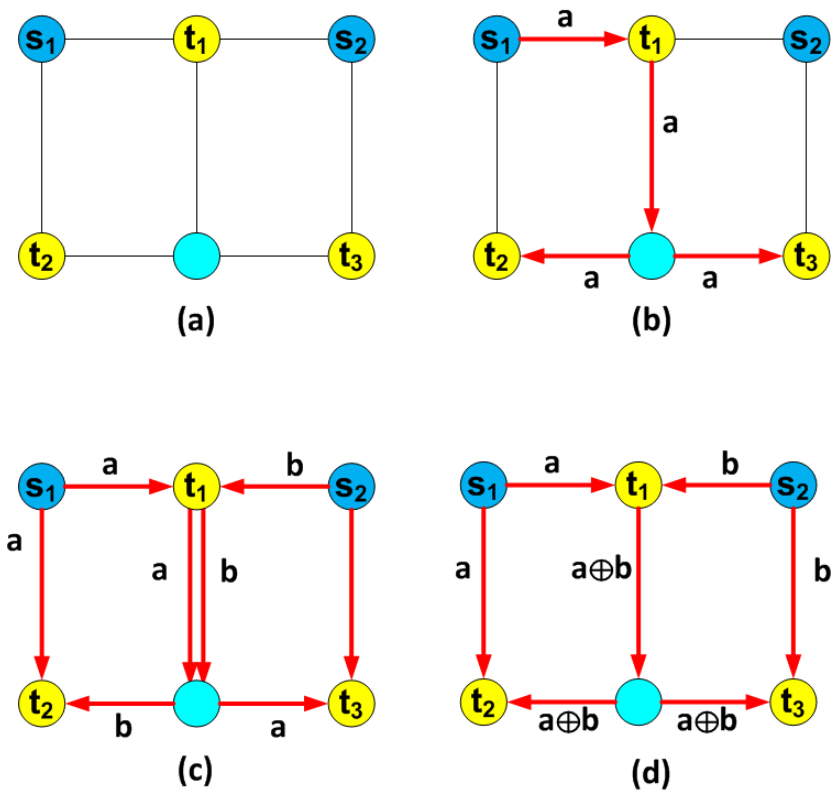

Fig. 10. (a) The underlying routing graph with a set of source nodes and terminals. (b) A rectilinear Steiner tree that connects source s_1 to all terminals. (c) A collection of two rectilinear Steiner trees that connect sources s_1 and s_2 to all terminals. (d) A network coding based solution.

CHAPTER II

EFFICIENT NETWORK CODING ALGORITHMS FOR DYNAMIC NETWORKS*

In this chapter we focus on design and analysis of efficient multicast network codes for dynamic networks. We present a distributed and fully deterministic algorithm for network code assignment that can deal with the dynamic nature of the network. Formally, we first consider the problem of maintaining the feasibility of a given network code upon a change in the network topology or addition of a new user. Our goal is to minimize the number of encoding coefficients that need to be modified to keep the network code feasible. We also present a new network coding algorithm that uses *path-based assignment* to efficiently handle frequent changes in the network topology and the multicast group.

A. Introduction

The central problem in multicast network coding is to design an efficient algorithm that assigns the encoding coefficients to the pair of edges in a way that allows each terminal node to decode the required packets. Such an assignment of local encoding coefficients is called a *feasible network code*. Currently, the main applications of the network coding technique are in the areas of content-distribution networks [7], peer-to-peer networks [8], and wireless networks [9]. Such networks typically have highly dynamic topologies and a frequently changing set of users. The dynamic nature of

*Parts of this chapter are reprinted with permission from "Efficient Network Coding Algorithms for Dynamic Networks" by M. A. R. Chaudhry, S. Y. EL Rouayheb, and A. Sprintson, in the proceedings of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops (SECON Workshops '09), Rome, Italy 2009, pages 1-6.

the networks arises because of many scenarios like varying capacity of the links due to congestion, hardware failures resulting in loss of communication links and addition of new communication links to compensate for broken links, and varying number of end users connected to a network. Any changes in the network might result in infeasibility of the previously assigned feasible network code.

In order to understand how dynamic network can affect the feasibility of a network code consider the following examples. Consider a mulitcast network with one source and four terminals $t_1,\ t_2,\ ,t_3,\ t_4$ as shown in Figure 11(a). The source wants to send two packets a and b to all the terminals. When a new terminal $\hat{t}$ joins the network and connects to nodes v_6 and v_8, the previously feasible network code, as shown in Figure 11(a), becomes infeasible as the new terminal is unable to decode the original packets sent by the source. For the same example, in case of a failure of an edge (v_2, v_7) as shown in 12(b) the effected terminals t_2 and t_3 respond to the loss of a link by connecting to nodes v_6 and v_8 respectively in order to maintain the minimum requirement. This change in topology makes the network code infeasible as shown in 12(b). Accordingly, there is a need to modify a number of network coding coefficients to make the network code feasible again. Keeping this modification to the minimum is practically important as it will allow the coding network to adjust for a new user or a change in the network topology more efficiently and also reduce the disruption for existing users.

1. Related Work

Network coding research has been initiated by the seminal paper by Ahlswede et al. [1], and since then attracted a significant interest from the research community. Koetter and Médard [10] developed an algebraic framework for network coding. This

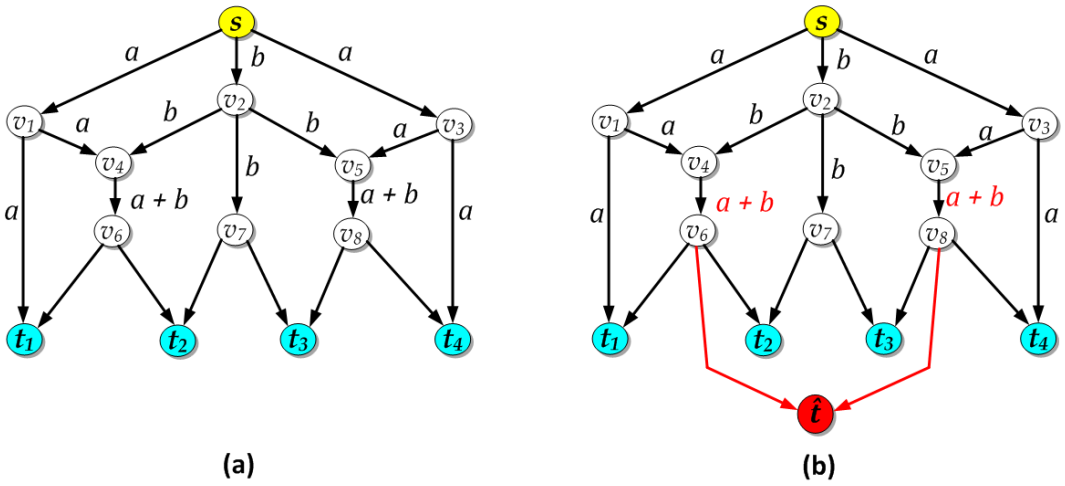

Fig. 11. (a) A network and corresponding feasible network code. (b) A new network $\hat{\mathbb{N}}$ after a new node $\hat{t}$ joins.

framework was used by Ho et al [11] to show that linear network codes can be efficiently constructed through a randomized algorithm. Jaggi et al [12] proposed a deterministic polynomial-time algorithm for finding feasible network codes in multicast networks. Network coding algorithms for dynamic networks have been studied in references [13], [14], and [15]. Ho et al [13] showed that the network coding approach provides substantial benefits in dynamically varying environments. Zhao and Médard [14] considered the problem of modifying network topology in a way that minimizes the number of required code rearrangements. Their algorithm uses linear programming approach and relies on a cost function that penalizes edges whose addition might require a change in the network code. Ho *e*t al [15] presented a framework for network management based on the network coding approach and considered the problem of minimizing the number of network codes required for handling all single edge failures. In reference [16] presented an overview of different algorithms to assign network coding coefficients.

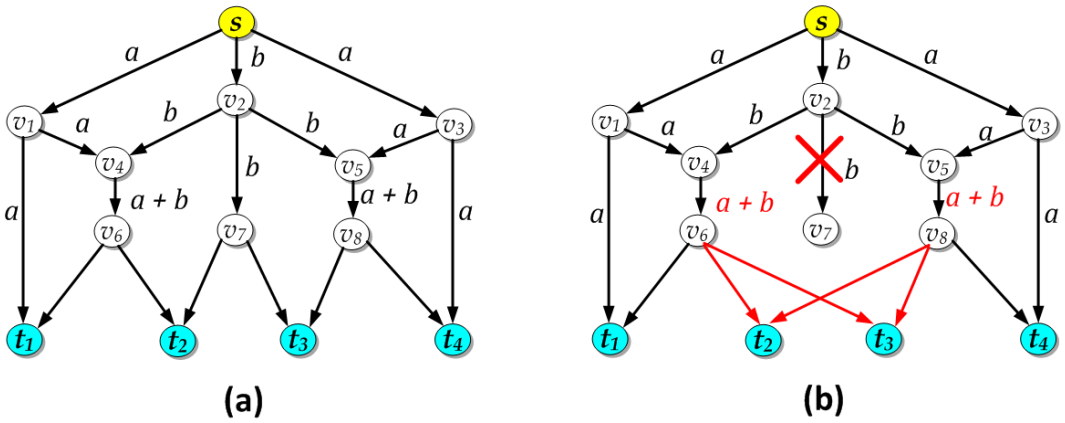

Fig. 12. (a) A network $\mathbb{N}$ and a feasible network code $\mathbb{C}$ for $\mathbb{N}$. (b) A new network $\hat{\mathbb{N}}$ after failure of an edge.

2. Organization

The rest of the chapter is organized as follows. In Section B we present the model and the definition of the problem. In Section C we analyze the problem in context of the changes in the multicast group due to the addition of a user. In Section D we analyze the problem in context of the changes in the topology due to failure of an edge. We present a new deterministic algorithm that uses path-based assignment of local encoding coefficients in Section E. Furthermore, in Section F we present a distributed, prime number's based assignment of encoding coefficient. In Section G we present the simulation study.

B. Model

We consider a multicast network $\mathbb{N}$ that uses a directed acyclic graph $G(V, E)$, with vertex set V and edge set E, to send data from source s to a set T of terminal nodes. The number of terminal nodes is denoted by k, i.e., $k = |T|$. The data is delivered in packets, each packet is an element of a finite field $\mathbb{F}_q = GF(q)$. We assume that

the communication is performed in rounds, such that each edge $e \in E$ transmits one packet per round. Note that this does not result in any loss of generality since edges of higher capacity can be substituted by multiple parallel edges. At each communication round, the source node needs to transmit h packets $\mathbb{R} = (p_1, p_2, \ldots, p_h)^T$ from the source node $s \in V$ to each terminal node $t \in T$. We refer to h as the *rate* of the multicast connection. It was shown in [1] and [17] that the maximum rate of the network, i.e., the maximum number of packets that can be sent from the source s to a set T of terminals per time unit, is equal to the minimum capacity of a cut that separates the source s from a terminal $t \in T$. Accordingly, we say that a multicast network $\mathbb{N}$ is *feasible* if any cut that separates s and a terminal $t \in T$ has at least h edges.

Without loss of generality, we assume that the source node s has exactly h incoming edges, each incoming edge transmits one of the original packets in $\mathbb{R}$. We also assume that each terminal $t \in T$ has h incoming edges and no outgoing edges. For each edge $e \in E$ we denote by p_e the packet transmitted on that edge.

Let $e(v, u)$ be an edge in E and let $\mathcal{M}_e$ be the set of incoming edges to its tail node v, $\mathcal{M}_e = \{(w, v) \mid (w, v) \in E)\}$. Then, we associate with each pair of edges $\{(e', e)|e' \in \mathcal{M}_e\}$ a *local encoding coefficient* $\beta_{e',e} \in \mathbb{F}_q$. The local encoding coefficients of the edges that belong to $\mathcal{M}_e$ determine the packet p_e transmitted on edge e as a function of packets transmitted on the incoming edges of e. Specifically, the packet p_e is equal to

$$p_e = \sum_{e' \in \mathcal{M}_e} \beta_{e',e} \cdot p_{e'}, \tag{2.1}$$

where all operations are performed over $\mathbb{F}_q$.

We say that edge e' is *adjacent* to edge e if the head node of e' is identical to the tail node of e. We denote by $\mathbb{S}$ the set of the adjacent pairs of edges in the network.

We refer to the set of local encoding coefficients $\mathbb{C} = \{\beta_{e',e} \mid (e', e) \in \mathbb{S}\}$ as a network code for $\mathbb{N}$.

Note that each packet transmitted over the network is a linear combination of the original packets $p_1, p_2, \ldots, p_h$ generated by the source node s. Accordingly, for each edge $e \in E$ we define the *global encoding vector* $\Gamma_e = (\gamma_1^e, \ldots, \gamma_h^e)^T \in \mathbb{F}_q^h$, that captures the relation between the packet p_e transmitted on edge e and the original packets in $\mathbb{R}$:

$$p_e = \sum_{i=1}^{h} \gamma_i^e \cdot p_i. \tag{2.2}$$

For each terminal t in T we define the *transfer matrix* $\mathbb{M}_t$ that captures the relation between the original packets $\mathbb{R}$ and the packets received by the terminal node $t \in T$ over its incoming edges. The matrix $\mathbb{M}_t$ is defined as follows:

$$\mathbb{M}_t = \left[\begin{array}{cccc} \Gamma_{e_t^1} | & \Gamma_{e_t^2} | & \ldots & | \Gamma_{e_t^h} \end{array} \right], \tag{2.3}$$

where $E_t = \{e_t^1, \ldots, e_t^h\}$ is the set of incoming edges of terminal t.

Our goal is to find a set of local encoding coefficients $\mathbb{C} = \{\beta_{e',e} \mid (e', e) \in \mathbb{S}\}$ that allows each terminal to decode the original packets $\mathbb{R}$ from the packets obtained on its incoming edges. This can be accomplished only if the matrix $\mathbb{M}_t$ is a full-rank matrix for each terminal $t \in T$. The assignment of $\mathbb{C}$ that satisfies this condition is referred to as *feasible network code*.

An edge $e(u, v)$ is referred to as a *forwarding edge* if it is an outgoing edge of the source node s or $p_e = \beta_{e',e} \cdot p_{e'}$ where $e' \in \mathcal{M}_e$ i.e., the packet p_e transmitted on edge e depends on only one of the incoming packets available at node u. Otherwise, edge e is referred to as an *encoding edge*. We say that a node v, $v \neq s$, is an *encoding node* if at least one of its outgoing edges (v, u) is encoding. If all outgoing edges of a node v are forwarding, then the node is referred to as a *forwarding node*. Encoding nodes

generate new packets by combining the packets received over their incoming edges; forwarding nodes only forward incoming packets. We denote by $\mathcal{X}(\mathcal{G}, \mathbb{C})$ the set of all encoding nodes in graph $G(V, E)$ with respect to a feasible network code $\mathbb{C}$.

Suppose that a change in network topology has occurred. We denote by $\hat{G}(\hat{V}, \hat{E})$ the new underlying graph, by $\hat{N}$ the new network, and by $\hat{S}$ the new set of adjacent edges. Let $\hat{C}$ be a feasible network code for $\hat{N}$.

We define following two types of changes associated with local encoding coefficients:

- We define set $\bar{S}$ that includes all pairs of adjacent edges for which the encoding coefficients have been changed as follows:

$$\bar{\mathbb{S}} = \left\{ (e', e) \mid (e', e) \in \mathbb{S} \cap \hat{\mathbb{S}}, \beta(e', e) \neq \hat{\beta}(e', e) \right\}.$$

  Note that $\bar{S}$ is the number of changed local encoding coefficients. Our goal is to minimize $|\bar{\mathbb{S}}|$.

- Total number of changes in encoding nodes is defined as:

$$|A \cup B|,$$

  where

$$A = \{v | v \in \mathcal{X}(\mathcal{G}, \mathbb{C}) \cap \mathcal{X}(\hat{\mathcal{G}}, \hat{\mathbb{C}}), (e'(u, v), e(v, w)) \in \mathbb{S} \cap \hat{\mathbb{S}}, \beta(e', e) \neq \hat{\beta}(e', e)\}$$

  and

$$B = \{v | v \notin \mathcal{X}(\mathcal{G}, \mathbb{C}), v \in \mathcal{X}(\hat{\mathcal{G}}, \hat{\mathbb{C}})\}$$

  In other words, by a change in an encoding node v we mean modifying $\beta_{e',e}$ for a pair of edges $e'(u, v)$ and $e(v, w)$, or modifying $\beta_{e,e'}$ for a pair of edges $e(u, v)$ and $e'(v, w)$ such that a non-encoding node v becomes an encoding node. We

refer to $|A \cup B|$ as the number of changes in encoding nodes required by $\hat{\mathbb{C}}$. Our focus is to minimize $|A \cup B|$. We refer to the problem to be the problem of finding a feasible network code $\hat{\mathbb{C}}$ that minimizes $|A \cup B|$ as the problem *EN*.

In this chapter we focus on network coding algorithms for dynamic networks. Specifically when a new user joins the network or when the network undergoes a topological change due to a failure of an edge. Upon addition of a new user or an edge failure the existing network code might no longer be feasible. A straightforward approach is to compute a new network code $\hat{\mathbb{C}}$ for $\hat{\mathbb{N}}$ from scratch. However, with this approach the local encoding coefficients for all pairs of edges in $\mathbb{S}$ might change. In practice, this may incur a substantial overhead, associated with determining, distributing, and changing the encoding coefficients for a large number of network nodes. However, in many cases, we only need to change a small number of encoding coefficients to make the new network feasible as shown in the following examples.

Example 1 (Adding a new terminal) *Consider network shown in Figure 13. The original network* $\mathbb{N}$ *together with the corresponding network code* $\mathbb{C}$ *is depicted in Figure 13(a). The network delivers two packets,* a *and* b*, to four terminals,* t_1, t_2, t_3*, and* t_4*. Suppose that a new terminal* $\hat{t}$ *joins the network. In order to maintain the feasibility requirement, two new edges,* $(v_6, \hat{t})$ *and* $(v_8, \hat{t})$ *have been added to the network. The resulting multicast network* $\hat{\mathbb{N}}$ *is depicted in Figure 13(b). Since the global encoding coefficients of edges* (v_4, v_6) *and* (v_5, v_8) *are linearly dependent, and since nodes* v_6 *and* v_7 *can only forward their incoming packets, we need to change some of the encoding coefficients for edges in* $\mathbb{S}$ *so the new terminal* $\hat{t}$ *will be able to decode the packets sent by source* s*. Figure 13(c) depicts a new network code* $\hat{\mathbb{C}}$ *formed from* $\mathbb{C}$ *by changing a single encoding coefficient* $(\beta_{(v_2,v_4),(v_4,v_6)})$*, i.e., the network code* $\hat{\mathbb{C}}$ *depicted in Figure 13 requires a change of only one coefficient.*

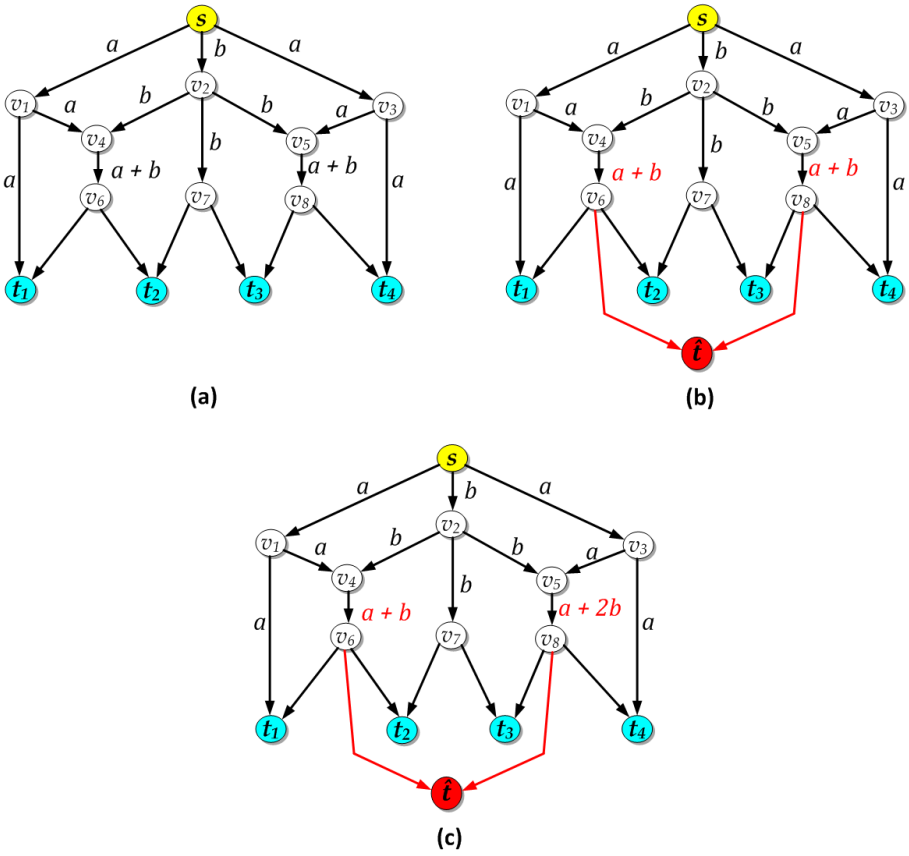

Fig. 13. (a) Original network $\mathbb{N}$ and a feasible network code $\mathbb{C}$ for $\mathbb{N}$. (b) A new network $\hat{\mathbb{N}}$ that includes a new terminal node $\hat{t}$. (c) A new network code $\hat{\mathbb{C}}$ for $\hat{\mathbb{N}}$.

Example 2 (Failure of an edge) *Consider the network* $\mathbb{N}$ *with the feasible network code* $\mathbb{C}$ *depicted in Figure 14(a), and assume that edge* (v_2, v_7) *has failed. The modified network topology is depicted in Figure 14(b). In the new topology, nodes* t_2 *and* t_3 *are connected to nodes* v_6 *and* v_8*. However, with the network code* $\mathbb{C}$*, the global encoding vectors of the edges* (v_4, v_6) *and* (v_4, v_8) *are linearly dependent. The new feasible network code* $\hat{\mathbb{C}}$ *for* $\hat{\mathbb{N}}$ *can be constructed by modifying the local encoding coefficient* $\beta_{(v_2,v_4),(v_4,v_6)}$ *for the pair of edges* (v_2, v_4) *and* (v_4, v_6)*, i.e., the network code* $\hat{\mathbb{C}}$ *depicted in Figure 14 requires a change of only one coefficient.*

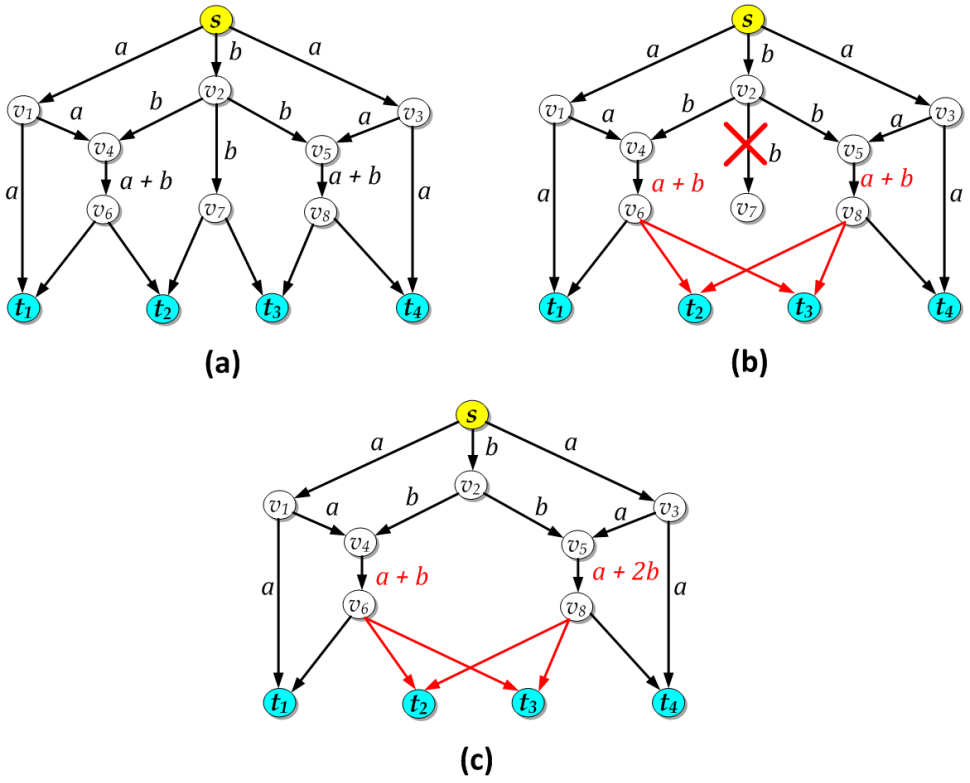

Fig. 14. (a) Original network $\mathbb{N}$ and a feasible network code $\mathbb{C}$ for $\mathbb{N}$. (b) A new network $\hat{\mathbb{N}}$ constructed after a failure of edge (v_2, v_7) with a new network code $\hat{\mathbb{C}}$ for $\hat{\mathbb{N}}$.

In what follows we discuss two specific scenarios of network under dynamic setting and describe the bounds on the number of changes required to maintain the feasibility of the code.

C. Adding a New Terminal to the Multicast Group

In this section we focus on the scenario in which a new terminal $\hat{t}$ joins the existing multicast group T. First, we establish lower and upper bounds on the number of modified network coding coefficients. Next, we prove that finding a modified network code with minimum number of changes in the encoding nodes is an NP-complete

problem.

Let $\mathbb{N}$ be the original multicast network over the graph $G(V, E)$, with a set of terminal nodes T and a corresponding feasible network code $\mathbb{C} = \{\beta_{e',e}\}$. Let $\hat{t}$ be a new terminal node, and $\hat{\mathbb{N}}$ be a modified multicast network over graph $\hat{G}(\hat{V}, \hat{E})$ and set of terminals $\hat{T} = \hat{t} \cup T$. We begin with the lower bound.

Lemma 3 *Let $\mathbb{M}_{\hat{t}}$ be the transfer matrix for $\hat{t}$ with respect to the code $\mathbb{C}$. Then, a feasible network code $\hat{\mathbb{C}}$ for $\hat{\mathbb{N}}$ requires at least $h - \text{rank}(\mathbb{M}_{\hat{t}})$ changes of the local encoding coefficients in $\mathbb{C}$, i.e., $|\bar{\mathbb{S}}| \geq h - \text{rank}(\mathbb{M}_{\hat{t}})$.*

Proof: It is sufficient to show that a change in one encoding coefficient can increase the rank of the transfer matrix by at most one. Suppose that we change the local encoding coefficient for a pair (e', e) of adjacent edges from $\beta'_{e',e}$ to $\beta''_{e',e}$. Let $\mathbb{M}'_{\hat{t}}$ and $\mathbb{M}''_{\hat{t}}$ be the transfer matrices before and after the change, respectively. Then $\mathbb{M}''_{\hat{t}}$ can be expressed as:

$$\mathbb{M}''_{\hat{t}} = \mathbb{M}'_{\hat{t}} + (\beta''_{e',e} - \beta'_{e',e})(\Gamma_{e'} \cdot \mathbb{M}_e), \tag{2.4}$$

where $\Gamma_{e'}$ is the global encoding coefficient for edge e' (before the change) and $\mathbb{M}_e$ is $1 \times h$ matrix that ties the packet sent on edge e and the packets received by the incoming edges of $\hat{t}$. Note that the rank of $\Gamma_{e'} \times \mathbb{M}_e$ is at most one. The subadditivity property of the rank function implies that the rank of $\mathbb{M}''_{\hat{t}}$ is bounded by the rank of $\mathbb{M}'_{\hat{t}}$ plus one. ■

We proceed to establish an upper bound.

Lemma 4 *Let $\hat{\mathcal{F}}$ be a set of h disjoint paths between s and $\hat{t}$ in $\hat{G}(\hat{V}, \hat{E})$. Let $\tilde{\mathbb{S}}$ be a set that contains pairs of adjacent edges in $\mathbb{S}$ such that each pair $(e', e) \in \tilde{\mathbb{S}}$ belongs to one of the paths in $\hat{\mathcal{F}}$. Then, a feasible network code can be constructed by changing the local encoding coefficients of the edges that belong to $\tilde{\mathbb{S}}$, i.e., $|\bar{\mathbb{S}}| \leq |\tilde{\mathbb{S}}|$.*

Note that the lemma implies that it is sufficient to only change $|\tilde{\mathbb{S}}|$ encoding coefficients.

Proof: First, we define a new network code $\hat{\mathbb{C}}$ as follows:

$$\hat{\beta}_{e',e} = \begin{cases} \beta_{e',e} + \Delta\beta_{e',e} & \text{if } (e',e) \in \mathbb{S} \text{ and } (e',e) \in \hat{\mathbb{P}} \\ \beta_{e',e} & \text{if } (e',e) \in \mathbb{S} \text{ and } (e',e) \notin \hat{\mathbb{P}} \\ \Delta\beta_{e',e} & \textit{otherwise.} \end{cases} \tag{2.5}$$

We show that it is possible to assign the values of $\{\Delta\beta_{e',e}\}$ such that the resulting network code $\hat{\mathbb{C}}$ is feasible. We show that for each $t \in \hat{T}$ the determinant $\det(\hat{\mathbb{M}}_t)$ of the transfer matrix $\hat{\mathbb{M}}_t$ is not identically equal to zero with respect to the new network code $\hat{\mathbb{C}}$. To this end, we substitute the values of coefficients in $\{\beta_{e',e}\}$ according to their assignment in $\mathbb{C}$ and leave $\{\Delta\beta_{e',e}\}$ to be variables. Then, for each $t \in \hat{T}$ the determinant of the transfer matrix $\hat{\mathbb{M}}_t$ is a multivariate polynomial in $\{\Delta\beta_{e',e}\}$. We observe that for each $t \in T$ this polynomial is not identically equal to zero. Indeed, with the assignment of $\Delta\beta_{e',e} = 0$ for each pair of adjacent edges (e',e) in $\hat{\mathcal{F}}$ the transfer matrix $\hat{\mathbb{M}}_t$ for each $t \in T$ is identical to the transfer matrix $\mathbb{M}_t$ for the same terminal under code $\mathbb{C}$. For terminal $\hat{t}$ the multivariate polynomial $\det(\mathbb{M}_{\hat{t}})$ will include an additive term $\prod_{(e',e)\in\hat{\mathcal{F}}} \Delta\beta_{e',e}$, hence this polynomial is also not identically equal to zero. Therefore, for a sufficiently large field ($q \geq |\hat{T}|$) it is possible to select the values of $\{\Delta\beta_{e',e}\}$ such that the transfer matrix for each terminal are invertible.

■

The modified network code can be constructed through a simple modification of the algorithm due to Jaggi et. al. [12]. In addition, a random algorithm for the network code assignment can be used.

Figure 15 shows an instance of the dynamic network coding problem for which the bound established by Lemma 4 is tight. The initial network includes nodes

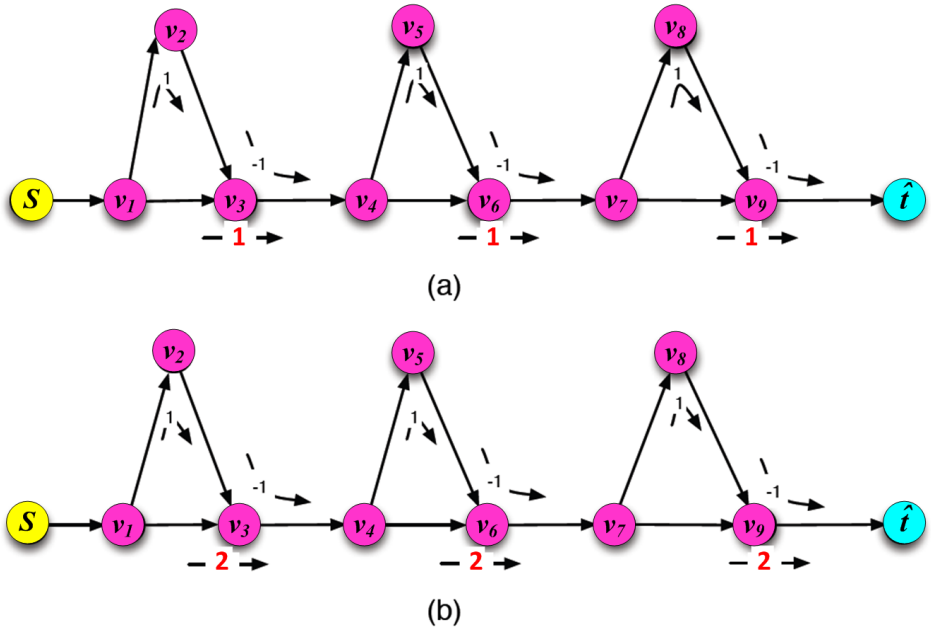

Fig. 15. An instance of a coding network. The arcs show local encoding coefficients between adjacent edges. (a) Original network code. (b) Modified network code.

$s, v_1, \ldots, v_9$. The existing network code $\mathbb{C}$ is shown in Figure 15(a). While the presented coding network is not minimal, all redundant edges can be justified by adding additional terminals. When a new terminal $\hat{t}$ joins the network, we need to modify at least three pairs of adjacent edges. Figure 15(b) shows a modified network code that requires at least three changes.

Next, we proceed to discuss the complexity of finding a feasible network code when a new terminal joins the network. The objective is to keep the number of changes in the encoding nodes to a minimum.

Theorem 5 *The problem* EN *is an NP-complete problem, i.e., finding a network code* $\hat{\mathbb{C}}$ *that requires the minimum number of changes in the encoding nodes is an*

NP-complete problem.

Proof:

We use a reduction from the Set Cover problem. The Set Cover problem is a known NP-hard problem [18]. The Set Cover problem is defined as follows. Given a universe set $U = \{x_1, \cdots, x_a\}$ and collection $C = \{c_1, \cdots, c_r\}$ of subsets of U, does there exist a subset of $C' \subseteq C$ of size k such that each element of U belongs to at least one member of C' (or if there exists a Set Cover of size k). Given an instance of the Set Cover problem we construct an instance of a dynamic network as follows:

- A dynamic network $\mathbb{N}$ that uses the graph $G(V, E)$ as shown in Figure 16. Each edge in $G(V, E)$ is of capacity a, except for dotted edges. Each dotted edge is of capacity 1. For each $c_i \in C$ there is node c_i in graph $G(V, E)$, and for each $x_i \in U$ there is a node x_i in $G(V, E)$. For each $x_j \in U$ and $c_i \in C$, such that $x_j \in c_i$, there is a dotted edge (c_i, x_j).Note that the capacity of minimum cut separating s from either of the terminals t_1 or t_2 is $r \cdot a$. So source s can send $r \cdot a$ packets to both the terminals t_1 and t_2.

- Initially terminal t_1 demands $r \cdot a$ packets from the source s as shown in Figure 16. Note, as s needs to transmit $r \cdot a$ packets to a single terminal t_1, so a flow of value $r \cdot a$ between s and t_1 is sufficient. So for all the pair of edges that are not the part of this flow we set their local encoding coefficients to be zero, and for rest of the edges encoding coefficient is chosen to be one, i.e., for each pair of edges (s^*, v_i), it holds that (v_i, u_i) $\beta_{(s^*,v_i),(v_i,u_i)} = 0$, and for each pair of edges (s, w_i), (w_i, u_i), it holds that $\beta_{(s,w_i),(w_i,u_i)} = 1$. Such a flow is shown in brown in Figure 16.

- Suppose that the terminal t_2 joins the terminal set.

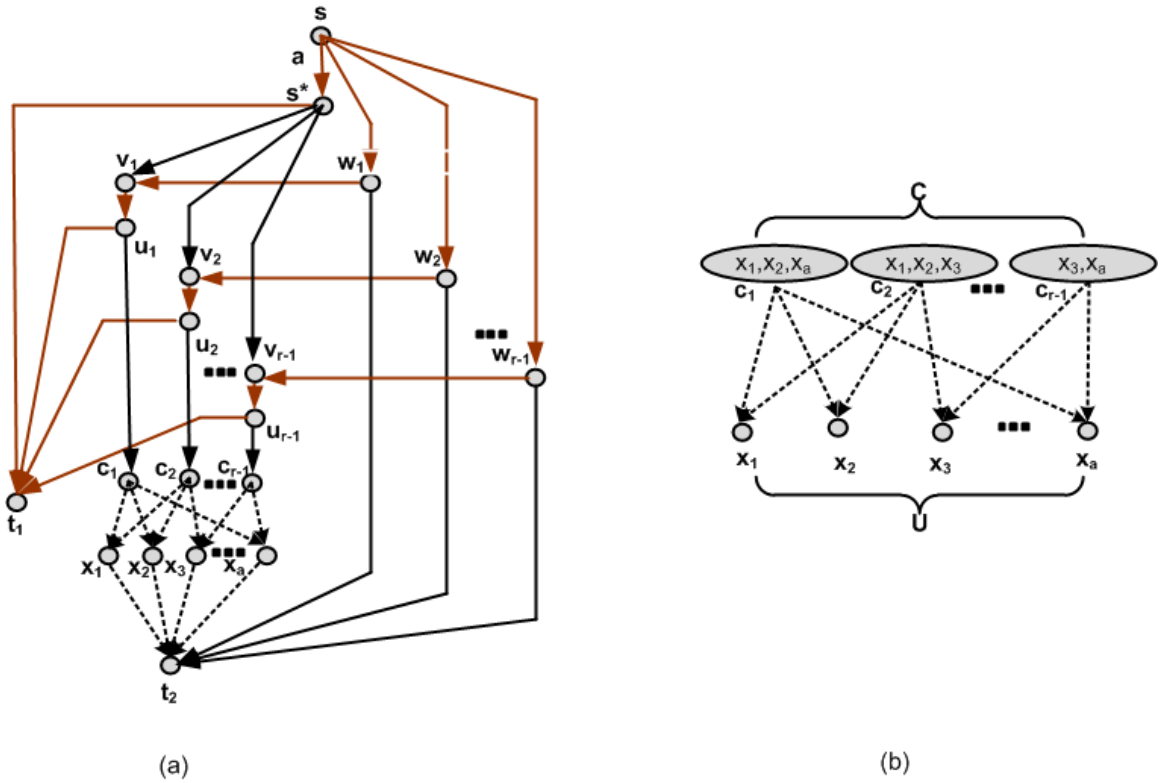

Fig. 16. (a) An instance of Set Cover problem (b) Corresponding Dynamic network $\mathbb{N}$.

- The objective is to satisfy the demands of both the terminals t_1 and t_2, while making least number of changes in the encoding nodes (i.e., already assigned encoding coefficients at the nodes).

We start by analyzing graph $G(V, E)$. First we note that the capacity of the minimum cut separating source s from either terminal t_1 or terminal t_2 is $r \cdot a$, so source s can send $r \cdot a$ packets to both of the terminals t_1 and t_2. Second, note that to satisfy both t_1 and t_2 source s must send a packets on each of edge (s, w_i), one packet on each of the dotted edges (x_i, t_2) , and a packets on the edge (s, s^*). Indeed removing (or decreasing capacity) of any of these edges reduces the minimum cut

capacity between s and t_1, or s and t_2. This also implies that the terminals t_1 and t_2 shall be getting $a \cdot (r-1)$ units of flow via edges (s, w_i), and terminal t_1 shall be getting the remaining a units of flow via the edge (s^*, t_1). So there is no other option for terminal t_2 to get the remaining a units of flow unless the packets sent on edge (s, s^*) are mixed with some of the packets sent on edges (w_i, v_i); each such mixing results in a change in an encoding node v_i (i.e., change of at least one local encoding coefficient from zero to some non-zero value for the pair of edges (s^*, v_i),(v_i, u_i)). Third note, as terminal t_2 must get a total of a units of flow from the edges (x_i, t_2) i.e., use all of the (x_i, t_2) edges, so the set of nodes c_i that are feeding these (x_i, t_2) edges correspond to set cover in the given instance of the Set Cover problem. This follows from the fact that an edge (c_i, x_j) exists if $x_j \in c_i$, and the union of the c_i feeding edges (x_i, t_2) is of cardinality a. So a feasible network code that meets the demands of both the terminals t_1 and t_2 results in a feasible solution to the Set Cover problem.

Four, we show that if there exists a subset of $C' \subseteq C$ of size k such that each element of U belongs to at least one member of C', then source can satisfy both the terminals t_1 and t_2 by making k changes in the encoding nodes. This follows from the fact that for all the packets transmissted on the edges (u_i, c_j) the only packets that might be useful for terminal t_2 are the ones that are mixed with packets sent on (s^*, w_i). Furthermore, mixing at an encoding node v_i corresponds to addition of c_i in C' in the Set Cover problem. So following point three, mentioned previously, if the demands of both the terminals can be met by making k changes in encoding nodes, then there exists a subset of $C' \subseteq C$ of size k such that each element of U belongs to at least one member of C'.

Hence the Set Cover problem can be reduced to problem *EN* i.e, the problem *EN* is an NP-hard problem. Furthermore, as a solution to the problem *EN* can be

verified in polynomial time, therefore the problem *EN* is an NP-complete problem.

■

D. Failure of an Edge

Dynamic networks can also experience frequent changes in the network topology. In particular, some of the network edges may fail and new edges and nodes may be added to the network to maintain the feasibility of the network. However under these changes the existing network code might not be feasible. Therefore to maintain the feasibility of network code some of the coding coefficients are required to be changed.

Let $\mathbb{N}$ be an original multicast network over the graph $G(V, E)$, with a set of terminal nodes T and a corresponding feasible network code $\mathbb{C} = \{\beta_{e',e}\}$. Let $\mathcal{F}_i$ be a set of h $\{P_1^i, \cdots, P_h^i\}$ disjoint paths between source s and terminal t_i. Let $\hat{e} \in E$ be a failed edge, and let $\hat{T} \subset T$ be effected terminals i.e. $\forall t_i \in \hat{T}$ the failed edge $\hat{e}$ belongs to $\mathcal{F}_i$. Let graph $\hat{G}(\hat{V}, \hat{E})$ be the graph formed from $G(V, E)$ by deleting edge $\hat{e}$. We begin by establishing the lower bound on the number of changes in encoding coefficients.

Lemma 6 *Let $\mathbb{M}_i$ be the transfer matrix for $t_i \in \hat{T}$ with respect to the code $\mathbb{C}$. Then, a feasible network code $\hat{\mathbb{C}}$ for $\hat{\mathbb{N}}$ over modified graph $\hat{G}(\hat{V}, \hat{E})$ requires at least $\max_{t_i \in \hat{T}}(h - \text{rank}(\mathbb{M}_i))$ changes of the local encoding coefficients in $\mathbb{C}$, i.e., $|\bar{\mathbb{S}}| \geq \max_{t_i \in \hat{T}}(h - \text{rank}(\mathbb{M}_i))$.*

Proof of lemma 6 is similar to that of Lemma 3. Now we proceed to establish an upper bound on the number of modified network coding coefficients required when an edge fails.

Lemma 7 *Let $\tilde{\mathcal{P}}_i$ be a set of disjoint path between source s and terminal $t_i \in \hat{T}$ such that failed edge $\hat{e} \in \tilde{\mathcal{P}}_i$. Let $\tilde{\mathcal{S}}_i$ be the set that contains the pairs of adjacent edges*

$(e', e) \in \tilde{\mathcal{P}}_i$. *Then, a feasible network code can be constructed by changing the local coding coefficients of edges that belong to union of* $\tilde{\mathcal{S}}_i$ *over all* $t_i \in \hat{T}$, *i.e.,* $|\bar{\mathbb{S}}| \leq \bigcup_{t_i \in \hat{T}} \tilde{\mathcal{S}}_i$.

Lemma 7 can be proven using similar arguments as given in proof of Lemma 4.

Next, we proceed to discuss the complexity of finding a feasible network code when an edge that has been transmitting packets fails, and the objective is to keep the number of changes in the encoding nodes to a minimum while finding a new feasible network code $\hat{\mathbb{C}}$.

Theorem 8 *Finding a network code* $\hat{\mathbb{C}}$ *that requires the minimum number of changes in the encoding nodes is an NP-complete problem.*

Proof:

We use reduction from the Edge-Disjoint-Path problem. The Edge-Disjoint-Path problem is defined as follows. Given a directed graph $G'(V', E')$ and two pairs of nodes $(\bar{s_1}, \bar{t_1}), (\bar{s_2}, \bar{t_2})$, $\bar{s_1}, \bar{t_1}, \bar{s_2}, \bar{t_2} \in V'$, do there exist two edge-disjoint paths in $G'(V', E')$ one connecting $\bar{s_1}$ to $\bar{t_1}$ and the other connecting $\bar{s_2}$ to $\bar{t_2}$. This problem is known to be NP-hard [19].

Given an instance of the Edge-Disjoint-Path problem we construct an instance of a dynamic network $\mathbb{N}$ that uses the graph $G(V, E)$ as shown in Figure 17. Graph $G(V, E)$ is formed by graph $G'(V', E')$ by adding nodes S, u,s_1,s_2,t_1, and t_2. The capacity of each edge in $G(V, E)$ is one. In the network $\mathbb{N}$ each of terminals t_1 and t_2 demands two packets from the source s. Initially source s transmits two packets to each t_1 and t_2 using the paths shown by dotted lines.

Suppose that, edge (s, u) fails. Then, based on the structure of the graph $G(V, E)$ to satisfy the demands of both the clients the source S send data through edges $(s_1, \bar{s_1})$, $(s_2, \bar{s_2})$, $(\bar{t_2}, t_1)$ and $(\bar{t_1}, t_2)$. The only way S can satisfy both clients without making any changes to the encoding nodes is to find two edge-disjoint paths, one

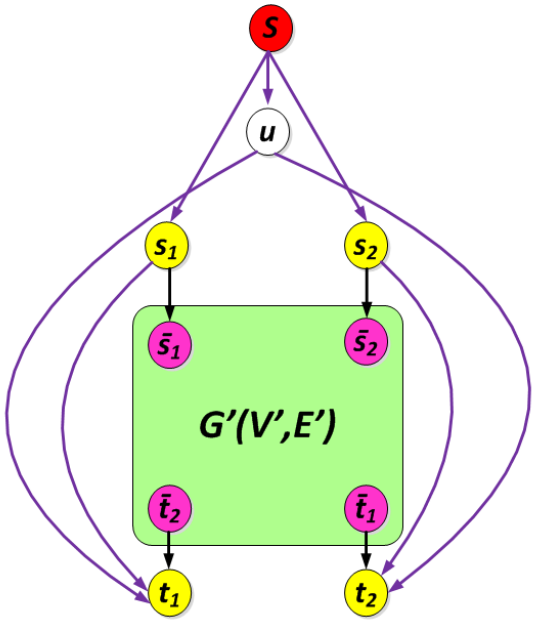

Fig. 17. Reduction from the Edge-Disjoint-Path problem to the Problem *EN*.

connecting $\bar{s_1}$ to $\bar{t_1}$ and the other connecting $\bar{s_2}$ to $\bar{t_2}$. Hence, if source S can satisfy the demands of both the terminals t_1 and t_2 without making any change to encoding nodes then there exist two disjoint paths in $G'(V', E')$ one connecting $\bar{s_1}$ to $\bar{t_1}$ and the other connecting $\bar{s_2}$ to $\bar{t_2}$, otherwise these paths do not exist. ■

E. Path-based Assignment of Encoding Coefficients

In this section we present a path-based approach for the assignment of the local encoding coefficients. We first present our approach in the context of static networks and then discuss its operation in dynamic networks.

Let $\mathbb{N}$ be the original multicast network that needs to deliver h packets per communication round from source node s to a set of terminal nodes $t_i \in T$ over communication graph $G(V, E)$. The first step of our algorithm is to determine, for

Algorithm *PBA* (G, s, T)

1 Determine, for each terminal $t_i \in T$, a set $\mathcal{F}_i$ of h edge-disjoint paths $\{P_1^i, \cdots, P_h^i\}$ between s and t_i;
2 Associate each terminal t_i with the encoding parameter φ_i;
3 For each pair of edges (e', e), where $e' = (u, v)$ and $e = (v, w)$ do;
4 Identify $T_{(e',e)} \subseteq T$ such that $\forall t_i \in T_{(e',e)}$ $(e', e) \in \mathcal{F}_i$;
5 Assign coding coefficients $\beta_{e',e}$ as follows:

$$\beta_{e',e} = \sum_{t_i | t_i \in T_{(e',e)}} \varphi_{t_i}; \tag{2.6}$$

Fig. 18. Algorithm *PBA*

each terminal $t_i \in T$, a set $\mathcal{F}_i = \{P_1^i, \ldots, P_h^i\}$ of h edge-disjoint paths between s and t_i. Then, we associate each terminal t_i with the encoding parameter φ_i. For every two pairs of adjacent edges $e' = (u, v)$, $e = (v, w)$ we define a subset $T_{(e',e)}$ of T which includes all terminals $t_i \in T$ for which it holds that both $e' = (u, v)$ and $e = (v, w)$ belong to the same path in $\mathcal{F}_i$. Then, the local encoding coefficient $\beta_{e',e}$ is defined to be the sum of the encoding parameters φ_i that correspond to the terminals in $T_{(e',e)}$. The formal description of the Algorithm *PBA* for path-based assignment of coding coefficients is given in Figure 18.

We demonstrate our approach through the following example. Consider the communication network presented in Figure 19(a). The network includes a source node s and a set of terminals $T = \{t_1, t_2, t_3\}$. First, we identify three disjoint paths to each terminal in T (see Figures 19(b)-(d)). Then, we associate each terminal t_i with an encoding parameter φ_i. Then, for each pair of adjacent edges e' and e we assign the corresponding local encoding coefficient $\beta_{e',e}$ to be the sum of the coefficients that correspond to the terminals which include both edges (v, u) and (u, w) on one of their disjoint paths from the source. Figure 19(e) shows the local encoding coefficients

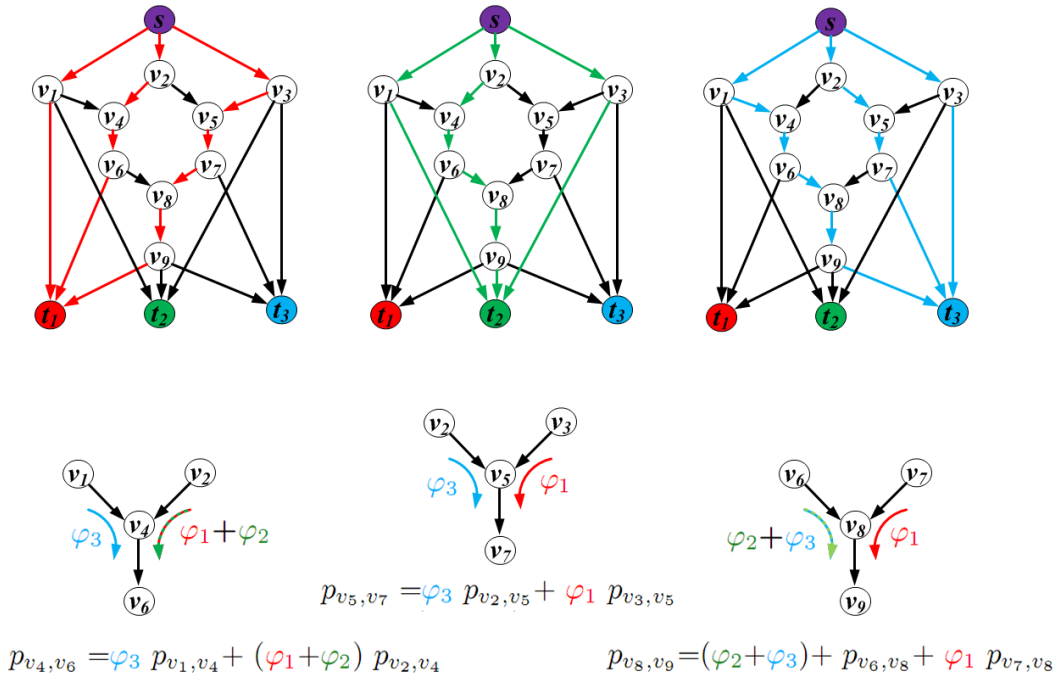

Fig. 19. (a) An instance to the network coding problem with three terminals t_1, t_2, and t_3; (b) Three disjoint paths to terminal t_1; (c) Three disjoint paths to terminal t_2; (d) Three disjoint paths to terminal t_3; (e) Local encoding coefficients for node v_4, v_5 and v_8. For example the packet $p_{(v_4,v_6)}$ sent on edge (v_4, v_6) is equal to $p_{(v_4,v_6)} = \varphi_3 \cdot p_{(v_1,v_4)} + (\varphi_1 + \varphi_2) \cdot p_{(v_2,v_4)}$.

assigned by our scheme for the network depicted in Figure 19(a). Our goal is to select the values of φ_i that yield a feasible network code. For the network depicted in Figure 19(a) it is easy to verify that $\varphi_1 = \varphi_2 = \varphi_3 = 1$ yields a feasible solution over $\mathbb{F}_3$.

Using the arguments similar to that used in Lemma 4, we can show that for a sufficiently large field size $\mathbb{F}_q$ there always exists a feasible assignment of the encoding coefficients. Moreover, when a new terminal $\hat{t}$ joins the network, it is possible to find a feasible value of $\varphi_{\hat{t}}$ provided that the total number of active terminals is bounded. However, finding a feasible assignments of $\varphi_{\hat{t}}$ requires full knowledge of the network

topology. In what follows we present an assignment of the encoding coefficients that does not require full knowledge.

F. Codes Based on Prime Numbers

The first step of the algorithm is to find a set $\mathcal{F}_i = \{P_1^i, \ldots, P_h^i\}$ of h edge-disjoint paths between s and t_i . Let θ_j denote the j'th prime number (i.e., $\theta_1 = 2, \theta_2 = 3, \theta_3 = 5, \ldots$). Let π_k be the product of the first k numbers, i.e.,

$$\pi_k = \prod_{j=1}^{k} \theta_j.$$

The steps for assigning coding coefficients are the same as given in Algorithm *PBA* except for the Step 2 where encoding parameter φ_i for terminal t_i is defined as follows:

$$\varphi_i = \pi_k/\theta_i = \prod_{j=1, j\neq i}^{k} \theta_j. \tag{2.7}$$

It is easy to see that this algorithm is distributed and does not require the full knowledge of network topology to assign local encoding coefficents. Indeed an encoding node only needs to know the flow passing through it to assign local encoding coefficients.

Figure 20 shows an example of a network code based on prime numbers for a network with two terminals.

In Theorem 9 below we show that the assignment of the local encoding coefficients given by Equation (2.7) is feasible over a sufficiently large primary finite field.

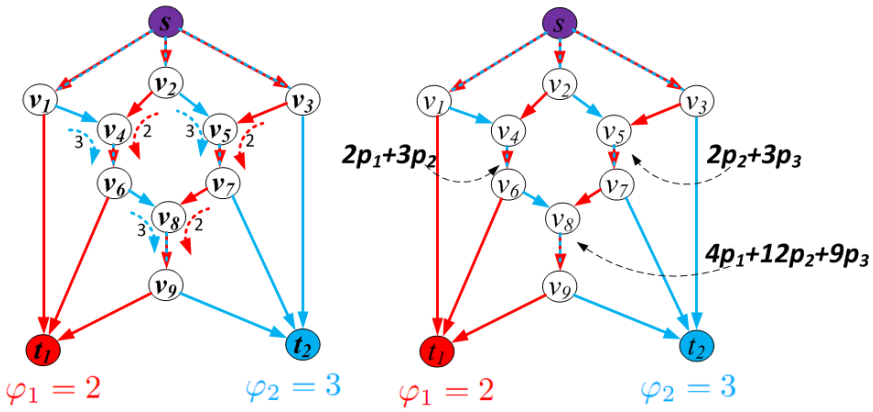

Fig. 20. (a) Local encoding coefficients, $\varphi_1 = \theta_2 = 3$ and $\varphi_2 = \theta_1 = 2$. (b) Global encoding coefficients.

Theorem 9 *Let $\mathbb{F}_q$ be a finite field such that*

$$q > (2 \cdot \pi_k)^{|E|},$$

where $|E|$ is the number of edges in a network. Then, the path-based assignment of the local encoding coefficients over $\mathbb{F}_q$ given by Equation (2.7) results in a feasible network code.

Proof: Let $\mathbb{N}$ be the coding network that needs to deliver h packets per communication round from source node s to a set of terminal nodes $t_i \in T$ over communication graph $G(V, E)$ and let $\{\beta_{e',e}\}$ be a set of local encoding coefficients for the adjacent edges. Let $\mathcal{F}_i$ denote a flow composed of h edge-disjoint paths $\{P_1, \ldots, P_h\}$ between s and a terminal $t_i \in T$. We say that a pair (e', e) of adjacent edges belongs to $\mathcal{F}_i$ if there exists a path P_i in $\mathcal{F}_i$ such that both edges, e' and e belong to P_i. We denote by

$$g(\mathcal{F}_i) = \prod_{\substack{(e',e) \text{ adjacent,} \\ (e',e) \in \mathcal{F}_i}} \beta_{e',e}$$

the *gain* of $\mathcal{F}_i$. Then, based on the result in [20, Th. 3], the network code $\{\beta_{e',e}\}$ is feasible, if and only if, for each terminal $t_i \in T$ it holds that

$$B_i = \sum_{\mathcal{F}_i : \mathcal{F}_i \text{ is a flow from } s \text{ to } t_i} \ell(\mathcal{F}_i) \cdot g(\mathcal{F}_i) \neq 0, \tag{2.8}$$

where $\ell(\mathcal{F}_i) \in \{1, -1\}$, and the summation is done over all flows between s and t_i.

Next, we will show that the choice of the φ_i's as described in Eq. (2.7) will lead to $B_i \neq 0$, for all $t_i \in T$. For every flow $\mathcal{F}$ from s to the terminal node t_i with a non-zero gain, one of the two following conditions holds:

1. Flow $\mathcal{F}$ is equal to flow $\mathcal{F}_i$. In this case $g(\mathcal{F})$ will include an additive term $\varphi_i^{N'} = (\pi_k/\theta_i)^{N'}$ for some integer N'. It easy to verify that all other multiple terms in $g(\mathcal{F})$ will have multiplicative factor θ_i.

2. Flows $\mathcal{F}$ and $\mathcal{F}_i$ are not identical. In this case, there exist adjacent edges $e' = (u, v)$, $e = (v, w)$ such that both e' and e belong to a path in $\mathcal{F}$, but there is no path in $\mathcal{F}_i$ that includes both e' and e. In this case, every additive term in $g(\mathcal{F})$ is divisible by θ_i.

We conclude that B_i will include one additive term $(\pi_k/\theta_i)^{N'}$ for some integer N', while rest of the additive terms will include θ_i as a multiplicative factor. The Fundamental Theorem of Arithmetic implies that B_i is not equal to zero when addition is over the integers. We note that this also holds if the size q of the finite field is greater than B_i.

Next, we show that $(2 \cdot \pi_k)^{|E|}$ is an upper bound on B_i. This will imply that for any prime field $\mathbb{F}_q$ such that $q \geq (2 \cdot \pi_k)^{|E|}$, B_i has a non-zero value.

First, we observe that each encoding coefficient is bounded by π_k. Second, any flow $\mathcal{F}_i$ from s to t_i contains at most $|E|$ edges, so the maximal gain $g(\mathcal{F}_i)$ of $\mathcal{F}_i$ is at

most $(\pi_k)^{|E|}$. We also note that the number of (s, t_i) flows is bounded by $2^{|E|}$. Thus, by Equation (2.8), $(2\pi_k)^{|E|}$ is an upper bound on the value of B_i and the theorem follows. ■

Our algorithms requires a finite field of size $(2\pi_k)^{|E|}$. It can be shown that the product of k prime numbers is bounded by 4^k. Then, the required size of the field is equal to $2^{(2k+1)|E|}$. Each element in such field can be represented by $(2k+1) \cdot |E|$ bits. Hence, the required packet size is linear in the number of edges in the network and the number of terminals.

After finding a set $\mathcal{F}_i = \{P_1^i, \ldots, P_h^i\}$ of h edge-disjoint paths between s and each of $t_i \in T$, the flows between s and all k terminals can be found in time $O(kVE \log(V^2/E))$ [21]. The amount of computations required is equal to number of encoding coefficients $\{\beta_{(e',e)}\}$, which is $O(|E|)$. Hence the running time of the proposed algorithm is $O(|E| + kVElog(V^2/E))$.

G. Simulation Results

In order to evaluate performance of the proposed algorithms we have used the practical ISP topologies of backbone networks determined by the Rocketfuel project [22] and the network code described in section II.F. Number of the links and backbone routers in each ISP determined by the Rocketfuel project [22] are shown in Table I.

1. Experimental Setup

For the purpose of simulations the backbone ISP map is transformed into a graph where each backbone router is taken as a node and a link between any pair of backbone routers is presented by a bidirectional edge of unit capacity. For each experiment we randomly select a multicast topology. More formally we choose a source and a set

Table I. Number of Routers and Links for ISP Backbone Networks.

AS	ISP	Routers	Links
1221	Telstra	355	700
1239	Sprintlink	547	1600
1755	Ebone	163	300
2914	Verio	1018	2300
3257	Tiscali	276	400
3356	Level3	624	5300
3967	Exodus	338	800
7018	AT&T	733	2300

of terminals from the graph according to a uniform distribution. The source and terminals are chosen such that mincut for each terminal from the source is at least 2. Then we find h disjoint paths from the source to each terminal where h is minimum of the mincuts among all the terminals. Without loss of generality, we only consider the instances of network topologies for which there exists a network coding opportunity otherwise we discard that network and restart by selecting a new set of source and terminal nodes. A Network coding opportunity exists if a mixing of packets at at-least one node is required. We have conducted experiments for counting the number of encoding coefficients that need to be modified to keep the network code feasible network upon a change in the multicast group.

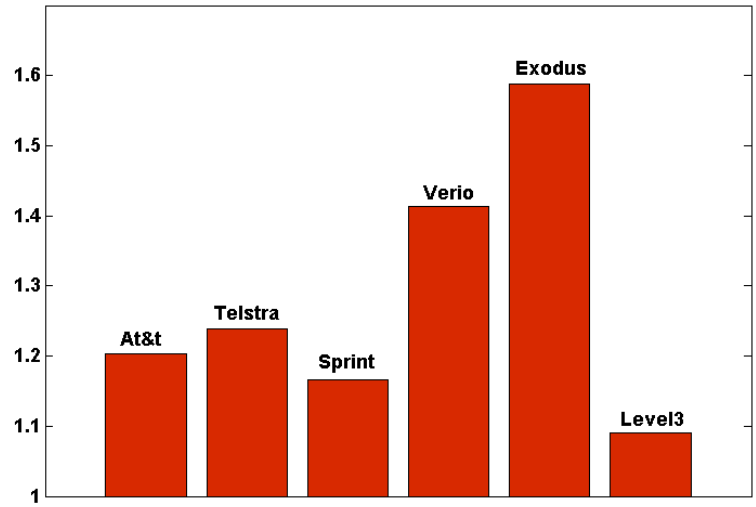

Fig. 21. Update Ratio for dynamic multicast networks.

2. Dynamic Networks

In this set of experiments we have compared proposed algorithm based on prime numbers given in section II.F with Deterministic Linear Information Flow (DLIF) algorithm proposed by Jaggi et al. [12]. The performance metric is termed as *update ratio*, which is defined to be the ratio of the "Number of update messages required by DLIF" to the "Number of update messages required by the proposed algorithm". Where number of update messages is equal to the number of encoding coefficients that need to be modified to keep the network code feasible upon change in the multicast group.

The proposed algorithm responds to changes in topology instantaneously because of its predetermined path coefficients, whereas in case of DLIF algorithm [12] a more extensive exercise for recomputing the number of updates messages is required.

Formally we start with a random multicast topology with a source and a set of 15 terminals. Then a random toss decides whether new terminal has to join or an

existing terminal has to leave. In case when a new terminal has to join a random node is selected from rest of the nodes (nodes that are neither source nor terminal) to join and in case of leaving, a random node from the existing nodes (source and terminals) is decided to leave the network. Next number of update messages required for this new multicast network is calculated. We have run 5000 such iterations for each ISP. Figure 21 shows the average update ratio required for each ISP backbone topologies over a set of 5000 experiments. The average value of update ratio is 1.86 which means that on average proposed algorithm requires 80% less update messages as compared to DLIF and this advantage can go as high as 172%. The results show that the proposed algorithm is capable of handling dynamic networks more efficiently.

H. Conclusion

This chapter focuses on minimizing the number of encoding coefficients that need to be modified to keep the network code feasible network upon change in the multicast group or a change in network topology. We have established an upper bound and a lower bound on the number of changes required in encoding coefficients under dynamic settings. We have also establish hardness results for the problem of minimizing the number of encoding coefficients in unaffected paths. To compute the encoding coefficients we came up with an efficient path-based algorithm . We have also presented an algorithm based on prime number which do not require full knowledge of network topology and is capable to handle needs of dynamic network more efficiently. The experimental study supports our claims

CHAPTER III

INDEX CODING*

In this chapter, we introduce the Index Coding problem and presents the model. The problem is motivated by several applications in wireless networking and distributed computing, including wireless architectures that utilize network coding and opportunistic listening. Upcoming chapters discuss efficient exact solution, several heuristic solutions, and approximation algorithms for special cases of the Index Coding problem.

A. Introduction

The *Index Coding* problem [4, 5] is one of the basic problems in the wireless network coding. Recently, it has attracted a significant attention from the research community (see e.g., [23–26] and references therein). An instance of the *Index Coding* problem comprises of a server, a set $X = \{c_1, \ldots, c_m\}$ of m wireless clients, and a set $P = \{p_1, \ldots, p_n\}$ of n packets that need to be delivered to the clients. Each client is interested in a certain subset of packets available at the server, and has a (different) subset of packets as *side information*. The server can transmit the packets to clients via a noiseless wireless channel. The goal is to find a transmission scheme that requires the minimum number μ of transmissions to satisfy the requests of all clients.

---

*Parts of this chapter are reprinted with permission from "Efficient Algorithms for Index Coding" by M. A. R. Chaudhry, and A. Sprintson, in the proceedings of the 2008 Annual IEEE International Conference on Computer Communications (INFOCOM) Workshops, Phoenix, U.S.A. 2008, pages 1-4.; and "On the Complementary Index Coding Problem" by M. A. R. Chaudhry, Z.Asad, A. Sprintson, and M. Langberg in the proceedings of the 2011 IEEE International Symposium on Information Theory (ISIT), St. Petersburg, Russia, 2011, pages 244-248.; and "Finding Sparse Solutions for the Index Coding Problem" by M. A. R. Chaudhry, Z.asad, A. Sprintson, and M. Langberg to appear in the proceedings of the 2011 IEEE Global Communications Conference (GLOBECOM), Houston, U.S.A., 2011.

Without loss of generality, we assume that the "wants" set of each client is of cardinality one. Indeed, if it is not the case, each client c_i whose "wants" set contains more than one packet can be substituted by multiple clients whose "wants" sets contain only one packet and whose "has" sets are equal to $H(c_i)$. We define the *coding gain* as the ratio between the minimum number of transmissions needed to satisfy all clients without encoding to the minimum number of transmissions required when encoding is used.

The Index Coding problem is motivated by various applications in wireless networking and distributed computing. For example, efficient index codes are instrumental for the wireless network architectures that utilize the network coding and opportunistic listening techniques [23, 27]. In addition, the Index Coding problem has several applications in satellite communication networks where the clients have limited storage and maintain part of the received information [5].

Figure 22 depicts a simple instance of the Index Coding problem with a server that needs to deliver five packets $P = \{p_1, \ldots, p_5\}$ to five clients. Each client requires a unique packet in P and has access to a subset of P. It can be verified that the demands of all clients can be satisfied by broadcasting three packets: p_1+p_2, p_3+p_4, and p_5 (all additions are over $GF(2)$). Note that with the traditional approach (without coding) all five packets $p_1, \ldots, p_5$ need to be transmitted.

We present our results in the context of wireless data transmission. However, the considered problem is very general and can arise in many other practical settings. For example, consider a content distribution network that needs to deliver several large files (such as video clips) to different clients. In this setting, if some of the files are already available to some clients then the distribution can be efficiently implemented by multicasting a (small) set of linear combinations of the original files. For example,

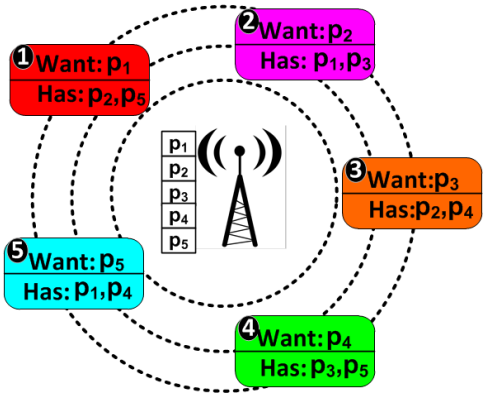

Fig. 22. An instance of the *Index Coding* problem.

Figure 23 depicts a video server that has three movies A, B, and C, and three clients that request one of the movies, while two other movies are already available to them. With the traditional approach, the server needs to transmit each movie separately to the client that has requested this movie. With the encoding approach the server only needs to multicast one file $A + B + C$, reducing the total amount of the consumed network resources and alleviating the congestion at the server.

The research on the Index Coding problem can be classified into two main directions. The first direction focuses on achievable rate bounds, as well as on the connections between the *Index Coding* problem and the *Network Coding* problem [28–30]. The second direction focuses on analyzing the computational complexity of the Index Coding problem as well as developing heuristic approaches to this problem [23–26,31]. In particular, the Index Coding problem has been shown to be NP-hard (in the linear setting). Moreover, finding an approximation solution for this problem is hard under a certain complexity assumption [31]. In this dissertation we have explored the later direction and present both the efficient heuristic based solutions as well as approximation solution for special cases of Index Coding problem.

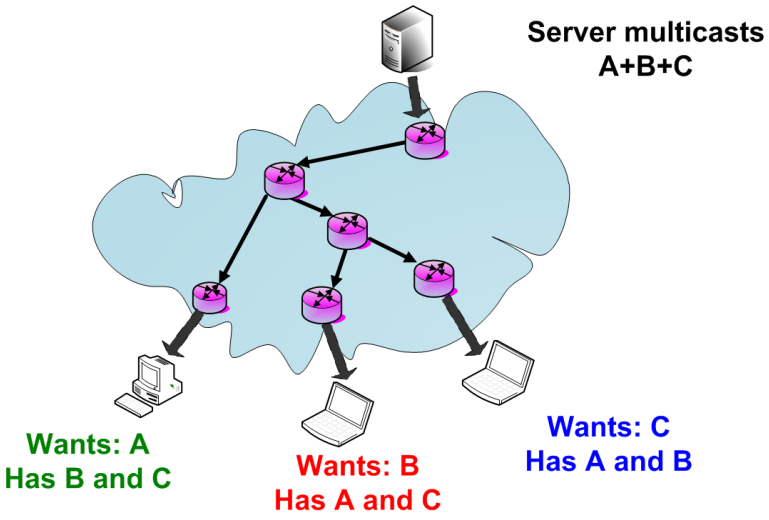

Fig. 23. A content distribution network.

B. Model

An instance of the *Index Coding (IC)* problem includes a server s, a set of m wireless clients $X = \{c_1, \ldots, c_m\}$, and a noiseless broadcast channel. The server holds a set of n packets, $P = \{p_1, \ldots, p_n\}$, that need to be transmitted to the clients. A client $c_i \in X$ is represented by a pair $(w_i, H(c_i))$, where $w_i \in P$ is the packet *required* by client c_i and $H(c_i) \subseteq P$ is the *side information* set available to client c_i.

Note that our assumption that each client requires a single packet does not limit the generality of the problem. Indeed, a client that requires more than one packet can be represented by multiple clients that share the same side information set, but require different packets.

We then define the *Complementary Index Coding (CIC)* problem where instead of minimizing the number of transmissions μ, our goal is to maximize the number of "saved" transmissions, i.e., $n - \mu$, where n is the number of packets that need to be delivered to the clients. Thus, the *CIC* problem seeks to maximize the benefit

obtained by employing the coding technique, e.g., for the problem instance shown in Figure 22, two transmissions can be saved by using coding. Note that, if OPT_{IC} is the optimum of the Index Coding problem and OPT_{CIC} is the optimum of the Complementary Index Coding problem, then it holds that $|OPT_{CIC}| = n - |OPT_{IC}|$.

In a *scalar-linear* solution, each packet is considered to be an element of the Galois field of order q, i.e., $p_i \in GF(q)$. A scalar-linear solution includes μ transmissions such that the packet $p^i = \sum_{j=1}^{n} g_i^j \cdot p_j$ transmitted at iteration i, $1 \leq i \leq \mu$ is a linear combination of packets in P, where $g_i = \{g_i^j\} \in GF(q)^n$ is the encoding vector for iteration i. To decode packet w_i, client c_i uses a linear decoding function r_i, such that $w_i = r_i(p^1, \ldots, p^\mu, H(c_i))$. The goal of the Index Coding problem is to find the minimum value of μ such that there exists a set of μ encoding vectors $g_1, g_2, \ldots, g_\mu$ and a set of m decoding functions $r_1, \ldots, r_m$ that allow each client to decode the required packet. The goal of the *CIC* problem is to maximize the value of $n - \mu$, i.e., the number of transmissions that were "saved" by using the encoding scheme.

In a *vector-linear* solution, each packet p_i is subdivided into k smaller size *subpackets* $p_i^1, \cdots, p_i^k$. Then, each transmitted packet is a linear combination of the subpackets $\{p_i^j |\ 1 \leq i \leq n, 1 \leq j \leq k\}$, rather than the original packets. With vector-linear network coding our goal is to find encoding and decoding schemes that minimize the ratio of $\frac{\mu}{k}$, where μ is the number of times a combination of subpackets is transmitted. For the *CIC* problem, the goal is to maximize the value of $n - \frac{\mu}{k}$.

For example, consider the setting depicted in Figure 22. As mentioned in the Introduction, the scalar-linear solution requires three transmissions (i.e., saves 2 transmissions). However, there exists a vector-linear solution that achieves $\frac{\mu}{k} = 2.5$ by dividing each packet into two parts (i.e., $k = 2$) and sending five combinations of subpackets, each subpacket is half the size of the original packet. Specifically, each packet p_i is divided into two sub-packets p_i^1, and p_i^2 and then the following five linear

combinations of the resulting packets are transmitted: $p_1^1+p_2^1$, $p_2^2+p_3^1$, $p_3^2+p_4^1$, $p_4^2+p_5^1$, and $p_5^2+p_1^2$.

Moreover, the *Sparse Index Coding (SIC)* problem is defined as a class of Index Coding problem where each transmitted packet must be a linear combination of *at most two* packets in P. For example consider an instance of the IC problem in which a server needs to deliver three packets p_1, p_2, p_3 to three clients. Each client *wants* one packet and *has* other two packets. An optimal solution to the IC problem can satisfy all clients by transmitting a single packet $p_1 + p_2 + p_3$. Note that in the case of the SIC problem where the server is restricted to encode at most two packets, the optimal scalar-linear solution can save 1 transmission by transmitting two packets $p_1 + p_2$, and $p_2 + p_3$. However in the case of the vector-linear solution to the SIC problem the optimal can save 1.5 transmissions by transmitting the packets $p_1^1 + p_2^1$, $p_2^2 + p_3^1$, and $p_3^2 + p_1^2$, where each packet p_i is divided into two half packets p_i^1, and p_i^2. We denote the minimum value of the scalar and vector-linear solutions to Problem SIC by OPT^s and OPT^f, respectively.

Wherever required, we shall introduce specific extensions or modifications of the model described here in Chapters IV, V and, VI where we deal with the special cases.

CHAPTER IV

EFFICIENT ALGORITHMS FOR INDEX CODING*

In this chapter, we propose efficient exact and heuristic solutions for Index Coding problem. Our numerical study shows that exact solutions can be efficiently obtained for small instances of the problem, while heuristic solutions with low computation time can achieve near-optimal performance for large instances.

A. Finding an Optimal Solution Through SAT Solver

Our exact solution is based on a reduction to the SAT problem which, in turn, can be efficiently solved by available SAT solvers such as *Chaff* [32] or *Minisat* [33]. In this section we assume that all the operations are performed over $GF(2)$.

In what follows we show how to check whether it is possible to satisfy all clients through k transmissions. The minimum value of k can be efficiently identified through the binary search algorithm. We need to check whether there exist k encoding vectors $g_1, \ldots, g_k$ of size n, and m decoding vectors $q_1, \ldots, q_m$ of size k that allow each client to decode the packet in its "wants" set. Here, g_j, $0 \leq j \leq k$ is the encoding vector used for transmission j, i.e., the packet x_j transmitted at round j is equal to $x_j = \sum_{t=1}^{n} g_j^t \cdot p_t$. Each client $c_i \in C$ uses a corresponding vector q_i to decode the packet in $W(c_i)$. In particular, it computes the following linear combination of the original packets:

$$\sum_{j=1}^{k} q_i^j \cdot x_j = \sum_{j=1}^{k} q_i^j \cdot \sum_{t=1}^{n} g_j^t \cdot p_t == \sum_{t=1}^{n} p_t \sum_{j=1}^{k} g_j^t \cdot q_i^j = \sum_{t=1}^{n} r_i^t p_t,$$

*Parts of this chapter are reprinted with permission from "Efficient Algorithms for Index Coding" by M. A. R. Chaudhry, and A. Sprintson, in the proceedings of the 2008 Annual IEEE International Conference on Computer Communications (INFOCOM) Workshops, Phoenix, U.S.A. 2008, pages 1-4.

where $r_i^t = \sum_{j=1}^{k} g_j^t \cdot q_i^j$ defines the linear combination of the original packets used by client c_i to decode the packet in $W(c_i)$. Note that client c_i will be able to decode the packet in $W(c_i)$ only if the following two conditions hold:

1. $r_i^t = 1$ for the packet p_t in the "wants" list $W(c_i)$ of c_i;
2. $r_i^t = 0$ for the every packet p_t that does not belong to either "has" list $H(c_i)$ of c_i or "wants" list $W(c_i)$ of c_i, i.e., $p_t \notin \{H(c_i) \cup W(c_i)\}$.

The two above conditions result in the following constraints on $\{g_i\}$ and $\{q_i\}$:

$$\sum_{j=1}^{k} g_j^t \cdot q_i^j \equiv 1 \quad \forall c_i \in C \text{ and } p_t \in W(c_i);$$

$$\sum_{j=1}^{k} g_j^t \cdot q_i^j \equiv 0 \quad \forall c_i \in C \text{ and } p_t \notin \{H(c_i) \cup W(c_i)\};$$

This, in turn, can be efficiently transformed into a Boolean problem by substituting the summation and multiplication over $GF(2)$ by the AND ($\wedge$) and XOR ($\oplus$) operations, respectively, over the boolean variables.

$$\oplus_{j=1}^{k} \left(g_j^t \wedge q_i^j\right) \equiv 1 \quad \forall c_i \in C \text{ and } p_t \in W(c_i); \tag{4.1}$$

$$\oplus_{j=1}^{k} \left(g_j^t \wedge q_i^j\right) \equiv 0 \quad \forall c_i \in C \text{ and } p_t \notin \{H(c_i) \cup W(c_i)\}. \tag{4.2}$$

While equations 4.1 and 4.2 can be easily transformed into the conjunctive normal form (CNF), a straightforward transformation may result in an exponential number of variables. Accordingly, in order to perform the transformation in an efficient manner, we use the Tstein transformation [34]. Such a transformation guarantees that the size of the CNF representation is linear in the size of equations (4.1) and (4.2).

Experimental Results: We have implemented the SAT approach and tested it on several random instances of the Index Coding problem. Specifically, given an instance of the original problem we transformed it to the CNF form and then invoked the *Minisat* [33] solver on the resulting CNF formula. Our results show that it is possible to efficiently obtain an optimal solution for instances that include up to 12 clients. For 10 clients we were able to achieve the coding gain as high as 2.4. Figure 24 shows the average coding gain as a function of the number of clients using the SAT-based solution.

B. Heuristic Approaches

The previous works [4, 35] show that it is NP-hard to find an optimal solution for the Index Coding problem. Moreover, it was shown in [31] that finding an approximate solution for this problem is also NP-hard. Accordingly, we present several heuristic approaches and compare their performance through simulations.

1. Reduction to Graph Coloring

The Index Coding problem is related to the graph coloring problem of an undirected graph $G(V, E)$. In graph coloring, we need to assign a color to each vertex $v \in V$ such that for any edge $(v, u) \in E$, the vertices v and u are assigned different colors, and the total number of colors is minimized. In this section we show a heuristic approach that transforms an instance I of the Index Coding problem to an instance I' of the graph coloring problem.

Specifically, consider an instance I of the Index Coding problem, in which the "wants" set of each client is of cardinality one. Then, we construct an instance $G(V, E)$ of graph coloring problem as follows:

- For each client $c_i \in C$ there is a corresponding vertex v_{c_i} in V;
- Each two vertices v_{c_i} and v_{c_j} are connected by an edge if one of the following holds:
  - Clients c_i and c_j have identical "wants" sets, i.e., $W(c_i) = W(c_j)$;
  - $W(c_i) \subseteq H(c_j)$ and $W(c_j) \subseteq H(c_i)$.

Let $\hat{V} \subseteq V$ be a clique in $G(V, E)$, i.e., each two vertices of $\hat{V}$ are connected by an edge in G. Note that all clients that correspond to nodes in $\hat{V}$ can be satisfied by one transmission, which includes a linear combination of all packets in their "wants" sets. Thus, we can minimize the number of transmissions by solving the *clique partition* problem [18]. In the clique partition problem, we need to partition set V into minimum number of disjoint subsets $V_1, V_2, \ldots, V_k$, such that for $1 \leq i \leq k$ the subgraph of G induced by nodes in V_i is a complete graph. Note that the clique partitioning problem for graph $G(V, E)$ is equivalent to the graph coloring problem of the complimentary graph $\hat{G}(V, \hat{E})$ of $G(V, E)$ (the complimentary graph $\hat{G}(V, \hat{E})$ of $G(V, E)$ contains all edges in $V \times V \setminus E$). The graph coloring problem for $\hat{G}(V, \hat{E})$ is a partition of V into minimum number of disjoint sets $V_1, V_2, \ldots, V_k$ such that each V_i is an independent set for $\hat{G}(V, \hat{E})$; $|V_1, V_2, \ldots, V_k| = \chi$ is called the *chromatic number of* $\hat{G}(V, \hat{E})$, or *minimum number of colors required to color* $\hat{G}(V, \hat{E})$. The graph coloring problem is a well-studied problem with a wealth of efficient heuristic solutions available in the literature. In general graphs no constant factor approximation algorithm exists for the graph coloring problem [36].

Our simulation results, shown in Figure 24 and Table II, show that this technique can be used for a larger number of clients compared to the optimal SAT-based solution although it is suboptimal in terms of the coding gains.

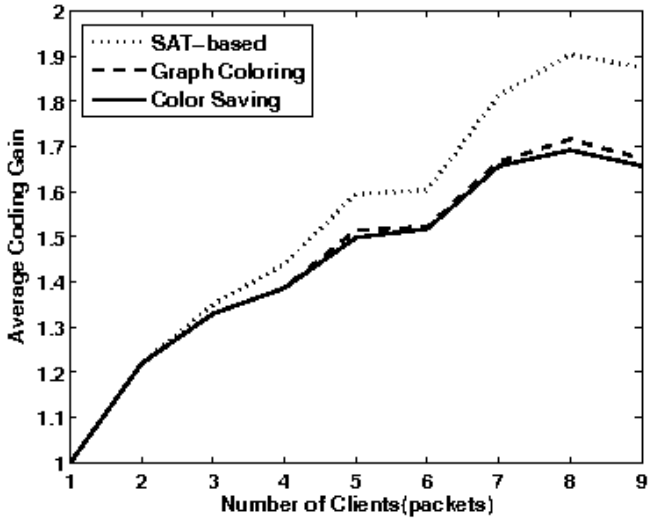

Fig. 24. Comparison of average coding gain between SAT-based, graph coloring and, color saving techniques.

2. Sparsest Set Clustering

In order to to be able to use our algorithm for a larger number of clients, we employ the "divide-and-conquer" approach. With this approach, the clients are divided into different groups, the problem is then solved for each group separately. The solution for the original problem is then combined from the solution of the subproblems.

The important decision in such an algorithm is to partition the clients into groups, in a way that minimizes the overall number of transmissions. We note that it is desirable to find a partition in which the clients that belong to different groups have in common as few packets in their "has" and "wants" sets as possible. For this purpose we construct an auxiliary directed graph $G(V, E)$ as follows:

- For each client $c_i \in C$ there is a corresponding vertex $v_{c_i} \in V$;
- Each two vertices v_{c_i} and v_{c_j} are connected by a directed edge (v_{c_i}, v_{c_j}) if one

of the following holds:

- Clients c_i and c_j have identical "wants" sets, i.e., $W(c_i) = W(c_j)$;
- $W(c_i) \subseteq H(c_j)$.

Then, our goal is to find a partition of $G(V, E)$ into two clusters of (almost) equal size such that the total number of edges that connect different clusters is minimized. To that end, we use the following greedy heuristic:

1. Partition the set V into two subsets V_1 and V_2 of (almost) equal size (in an arbitrary way);
2. If there exists a node $v \in V_1$ that has more neighbors in V_2 than in V_1, move v to V_2;
3. If there exists a node $v \in V_2$ that has more neighbors in V_1 than in V_2, move v to V_1.

This procedure continues until we have nodes in V_1 or V_2 that have more neighbors in the opposite cluster than the number of neighbors in their current cluster or the cluster of desired size is obtained. The resulting two subgraphs are recursively partitioned through the same procedure until all the size of each cluster is less than or equal to the desired value.

Figure 25 shows histograms of coding advantage for 100 experiments. In each experiment we generated 150 clients with random "has" and "wants" sets. Then, we divided the clients to subsets of at most seven clients each. For each subset, the optimal number of transmissions was found through the SAT-based approach described in Section IV.A.

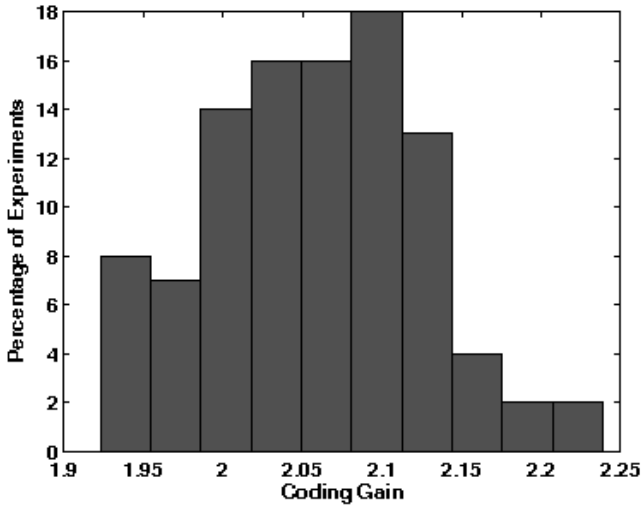

Fig. 25. Histogram of the coding gain values for 150 clients with 100 experiments using sparsest set clustering.

3. Using Color Saving Heuristic

This approach optimizes the number of transmissions saved by Index Coding, and results in constant factor approximation guarantee. In this section we are working on the same auxiliary graph $G(V, E)$ as as described in Section IV.B.2.

The heuristic we used is based on the approximation algorithms for maximizing the number of unused colors where the number of unused colors is the difference between the number of vertices in the graph and number of colors required to color the graph. Though graph coloring and color saving are very related problems, it is not possible to have a constant factor approximation solution for graph coloring, whereas constant factor approximation solutions exist for color saving [37].

Some of the notations used in this section are: Let χ be the minimum number of cliques required to cover graph $G(V, E)$ (an optimal solution to the *clique partition*

problem [18]), where $G(V, E)$ is the same auxiliary graph as described in Section IV.B.1.

We have used following two different techniques to solve the color saving.

a. Greedy Color Saving

The following greedy algorithm is based on approximation solution for the color saving due to Hassin et.al [38].

The algorithm performs the following steps:

1. Construct an undirected graph $G(V, E)$ as described in Section IV.B.1;
2. **While** there exists a clique of size 3 in $G(V, E)$ **do:**
   (a) Find a clique $\{v_i, v_j, v_k\}$ of size 3 in $G(V, E)$;
   (b) Create a packet that satisfies all clients in $\{v_i, v_j, v_k\}$;
   (c) $V := V \setminus \{v_i, v_j, v_k\}$;
3. Compute a maximum matching of $G(V, E)$;
4. For each pair $\{v_i, v_j\}$ in the matching create a packet that satisfies all clients in $\{v_i, v_j\}$;
5. Create a new packet for each one of the remaining vertices of V.

Our results show that this algorithm is very fast, can work with a very large number of clients and has a very good performance in terms of minimizing the overall number of transmissions. Figure 26 depicts the comparison of the color saving and sparsest set clustering techniques and shows that the color saving technique clearly outperforms sparsest set clustering, though for very large number of clients sparsest set clustering is the only option giving results in a reasonable amount of time.

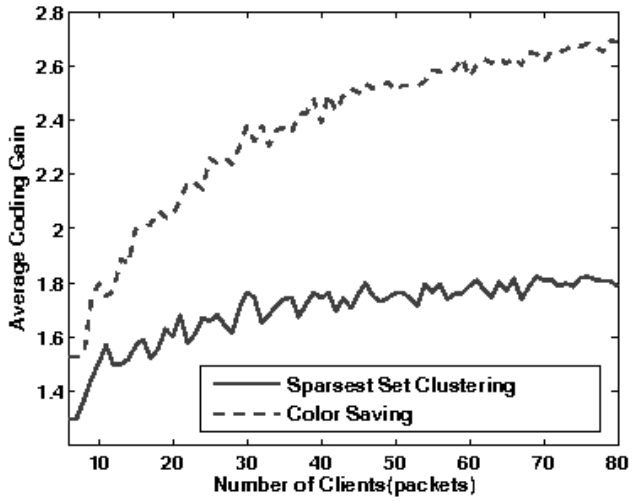

Fig. 26. Comparison of average coding gain between the sparsest set clustering and color caving techniques.

b. Reduction to Set Cover

The color saving problem can be reduced to the set cover problem. For solving the color saving problem using 3-set cover, the algorithm in [37] proposes an approximation solution which is guaranteed to be within a factor of $\frac{5}{6}$ of the optimal solution. The approximation uses the idea of semi-local optimization in which solution is improved iteratively in each step until no more improvements can be achieved. Formally in each step 3-sets is selected for partial cover and then the remainder of universe is covered optimally by computing maximum matching on 2-set and 1-sets of uncovered elements. A semi local(s,t)-improvement step which is carried out only on 3-sets means insertion of up to s 3-sets and deletion of up to t 3-sets. Quality of improvement is the measure of number of sets i.e. a cover with less number of sets is better and for the covers of same size the one with lesser number of 1-set is better.

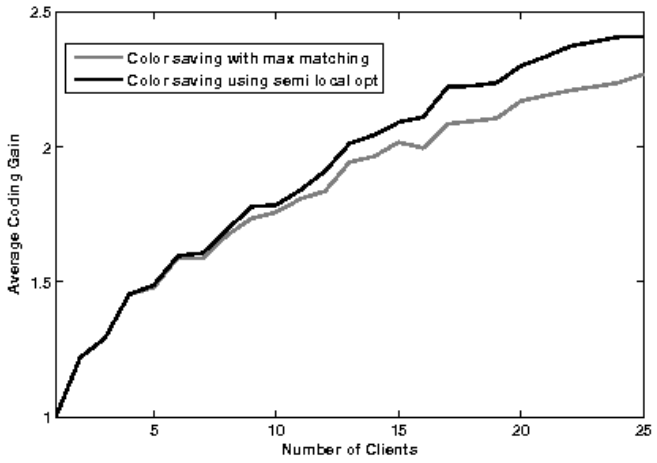

Fig. 27. Comparison of average coding gain between the greedy color saving technique and color saving using (2,1)semi-local optimization.

The authors in [37] show that best choice for (s, t) is (2,1).

The algorithm performs the following steps:

1. Construct an undirected graph $G(V, E)$ as described in Section IV.B.1;
2. Let universe U be the set of all nodes in the graph, i.e., $U = V$
3. Let the Set system S consist of all cliques of size at most 3 in graph G. Note that set system is monotone i.e. is set s is in S then s' is also in S, for $s' \subset s$
4. Find a greedy solution for the disjoint k-set cover
5. Find Set Cover using $(2, 1)$semi-local optimization given in [37]
6. Number of colors=|setCover|;

C. Numerical Results

Figure 24 depicts the comparison of the average coding gain for SAT-based, graph coloring and color saving solutions to the Index Coding problem. It shows that when the number of clients is very small all techniques yield almost same average coding gain as there are very few coding opportunities. However, if the number of clients is large, then the SAT-based solution (which is the optimal method) yields a much larger gain as compared to the other techniques. The graph coloring and color saving techniques yielded almost the same coding gain. We also observed that average coding gain increases with the number of clients. The comparison of average coding gain between the greedy color saving technique and color saving using (2,1)semi-local optimization is shown in Figure 27.

SAT-based solution can identify the solution efficiently for instances with up to nine clients. For instances with a larger number of clients, graph coloring is a more efficient technique as shown in Table III.

D. Conclusion

In this chapter we presented efficient solutions to the Index Coding problem. Our algorithms minimize the number of transmissions necessary for satisfying the requests of all clients. We presented an optimal solution using a SAT solver and several efficient heuristic solutions using the graph coloring, sparsest set clustering and color saving techniques and studied their performance through extensive simulations.

Table II. CPU Time (in Seconds) Required by SAT-based, Graph Coloring and Color Saving Techniques.

No. of Clients	**SAT-based**	**Graph Coloring**	**Color Saving**
2	0.63	0.03	0.00506
3	0.91	0.06	0.07654
4	0.7341	0.06942	0.01524
5	0.9391	0.07914	0.01492
6	0.9622	0.1043	0.0454
7	3.579	0.1618	0.1285
8	11.93	0.2637	0.09848
9	82.97	0.7654	0.05246

Table III. CPU Time (in Seconds) Required for Sparsest Set Clustering and Color Saving Techniques.

No. of Clients	**Sparsest Set Clustering**	**Color Saving**
6	0.5636	0.03013
10	0.7807	0.0552
20	1.587	0.14
40	3.28	0.514
80	6.764	5.829
100	9.375	15.719
160	15.656	143.188

CHAPTER V

COMPLEMENTARY INDEX CODING*

In this chapter, we consider a complementary problem whose goal is to maximize the number of *saved transmissions*, i.e., the number of transmissions that are saved by combining packets compared to the solution that does not involve coding. We refer to this problem as the *Complementary Index Coding* problem. It turns out that the complementary problem can be approximated in certain cases of practical importance.

We consider the *multiple unicast* and *multiple multicast* scenarios. In the multiple unicast scenario, each packet is requested by a single client; while in the multiple multicast scenario, each packet can be requested by several clients. For the multiple unicast scenario, we present approximation algorithms for finding *scalar* and *vector* linear solutions. For the multiple multicast scenario, we show that finding an approximation solution is NP-hard.

A. Introduction

In this part of dissertation we focus on the *Complementary Index Coding (CIC)* problem. In this problem, instead of minimizing the number of transmissions μ, our goal is to maximize the number of "saved" transmissions, i.e., $n - \mu$, where n is the number of packets that need to be delivered to the clients. Thus, the *CIC* problem seeks to maximize the benefit obtained by employing the coding technique, e.g., for the problem instance shown in Figure 22, two transmissions can be saved

*Parts of this chapter are reprinted with permission from "On the Complementary Index Coding Problem" by M. A. R. Chaudhry, Z. Asad, A. Sprintson, and M. Langberg in the proceedings of the 2011 IEEE International Symposium on Information Theory (ISIT), St. Petersburg, Russia, 2011, pages 244-248.

by using coding. Note that, if OPT_{IC} is the optimum of the Index Coding problem and OPT_{CIC} is the optimum of the Complementary Index Coding problem, then it holds that $|OPT_{CIC}| = n - |OPT_{IC}|$. This implies that the *CIC* problem is also NP-hard. However, as we show in this chapter, the *CIC* problem can be successfully approximated in many cases of practical importance.

There are several well-known optimization problems which are complementary to one another. Examples include the Vertex Cover and Independent Set problems [18] and the problems of coloring (finding the minimum chromatic number of a graph) and that of *color saving* (see e.g., [37]). In the context of approximation, complementary problems may have a significantly different behavior as a good approximation algorithm to one of the problems does not imply one for its companion (e.g., [39]).

In this chapter we study *scalar* and *vector* linear solutions for the *Complementary Index Coding* problem. In a vector-linear solution, each packet can be split into smaller sub-packets, such that sub-packets originated from different packets can be combined together. In the scalar-linear solution, the packets cannot be split. We consider two scenarios for *Complementary Index Coding*, i.e., the *Multiple Unicast* scenario and the *Multiple Multicast* scenario. In the multiple unicast scenario, each packet is requested by a single client; in the multiple multicast scenario, each packet can be requested by more than one client. For the multiple unicast scenario, we present approximation algorithms for finding scalar and vector-linear solutions that achieve approximation ratios of $\Omega(\sqrt{n} \cdot \log n \cdot \log \log n)$ and $\Omega(\log n \cdot \log \log n)$, respectively. For the multiple multicast scenario, we show that finding an approximation solution is NP-hard.

Our algorithms for multiple unicast strongly build on the notion of a *dependency graph* $G(V, E)$ that captures the combinatorial properties of the *IC* and *CIC* problems (the *dependency graph* $G(V, E)$ shall be defined later in this chapter and is slightly

different from the auxiliary graph defined in Section IV.2.). Loosely speaking, we show that any cycle in this graph G enables to save a single transmission. Thus, by finding many such disjoint cycles, via the algorithms presented in [40] and [41], one may save several transmission and obtain an approximation to Problem *CIC*. To quantify the amount of transmissions saved in this manner we tie the value of the optimal coding scheme to the *feedback vertex set* of $G(V, E)$. For multiple multicast, we show a reduction from the *CIC* problem to the Independent Set problem, implying hardness of approximation in this case.

The rest of this chapter is organized as follows. Section V.B introduces the notion of the *dependency graph* and related problems. Sections V.C and V.D focus on multiple unicast and multiple multicast cases, respectively. Section V.E presents the simulation study.

B. Preliminaries

We start by introducing a notion of the *Dependency Graph*. The dependency graph is defined for the multiple unicast case, in which each packet is requested by a single client.

Definition 10 (Dependency Graph) *Given a multiple unicast instance of the* CIC *problem we define a graph $G(V, E)$ as follows:*

- *For each client $c_i \in X$ there is a corresponding vertex v_{c_i} in V*
- *There is a directed edge from v_{c_i} to v_{c_j} if and only if it holds that $w_i \in H(c_j)$.*

Figure 28 depicts the dependency graph that corresponds to the instance of the *CIC* problem depicted in Figure 22.

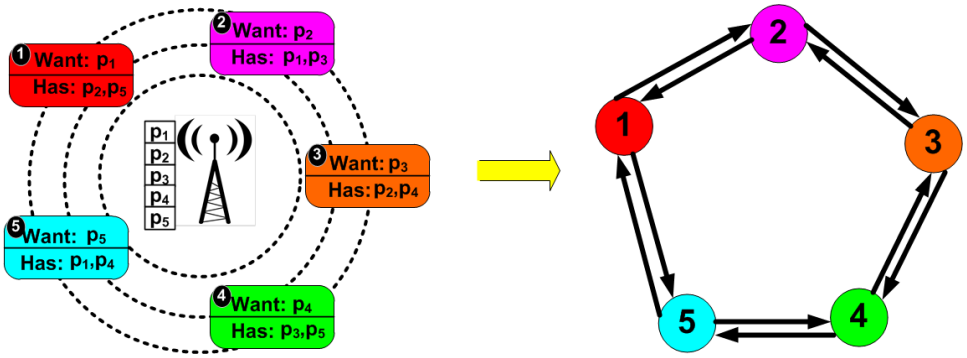

Fig. 28. Dependency graph for the instance of the *CIC* problem depicted on Figure 22.

We proceed to define the notion of the *Maximum Induced Acyclic Subgraph (MAIS)* [4].

Definition 11 (Maximum Induced Acyclic Subgraph (MAIS)) *For a given graph* $G(V, E)$*, we define* Maximum Induced Acyclic Subgraph *(*MAIS(G)*) as the maximum acyclic induced subgraph of* G*. We denote by* |MAIS(G)| *the number of vertices in* MAIS(G)*.*

It was shown in [4] that in the multiple unicast case the optimal solution to the Index Coding problem is larger or equal to MAIS(G).

Other related problems are *the minimum Feedback Vertex Set (FVS)* and *the minimum Feedback Edge Set (FES)* of a graph.

Definition 12 (Problems FVS and FES) *For a given graph* G*, find the least number of vertices (edges) whose removal makes the graph acyclic. We denote the optimal integral solution to the* FVS *(*FES*) problem for a graph* $G(V, E)$ *by* $|FVS(G)|$*,* $|FES(G)|$*.*

Note that, for a given graph G, by definition of the *Feedback Vertex Set*, it holds that $|MAIS(G)| = n - |FVS(G)|$.

Both the *FVS* and *FES* problems can be formulated as integer programs. By relaxing the integrality constraints on the decision variables, we can get fractional version of these problems. The fractional feedback vertex set ($FVS^f(G)$) for the graph $G(V, E)$ is a function $t : V \rightarrow [0, 1]$ such that every cycle, c, is covered by t, i.e., $\sum_{v \in c} t(v) \geq 1$. The fractional feedback edge set ($FES^f(G)$) for the graph $G(V, E)$ is a function $t : E \rightarrow [0, 1]$ such that every cycle, c, is covered by t, i.e., $\sum_{e \in c} t(e) \geq 1$. An optimum feedback vertex (edge) set has minimizes $\sum_{v \in V} t(v)$. The values of optimal fractional solutions to these problems are denoted by $|FVS^f(G)|$ and $|FES^f(G)|$, respectively.

Proposition 13 $|OPT_{CIC}| \leq |FVS(G)|$

Proof: In [4] it was shown that $|OPT_{IC}| \geq |MAIS(G)|$. Since $|MAIS(G)| = n - |FVS(G)|$, we get $n - |OPT_{IC}| \leq |FVS(G)|$. Furthermore, by the definition of the *CIC* problem $|OPT_{CIC}| = n - |OPT_{IC}|$. Hence, $|OPT_{CIC}| \leq |FVS(G)|$. ■

In our work we also use a notion of the *Vertex Split Graph*, defined as follows:

Definition 14 (Vertex Split Graph) *Given a graph $G(V, E)$ we construct a corresponding* Vertex Split Graph *$G'(V', E')$ as follows: (1) For each node $v \in V$: (a) create two nodes v_{in}, v_{out} in V' (b) create an edge (v_{in}, v_{out}) in E'; (2) For each edge $(u, v) \in E$ create an edge (u_{out}, v_{in}) in E'.*

Note that the size of the Feedback Vertex Set of G is equal to the size of Feedback Edge Set of its vertex split graph G', i.e., $|FVS(G)| = |FES(G')|$.

We proceed to describe Edge-disjoint Cycle Packing (ECP) and Vertex-Disjoint Cycle Packing (VCP) problems. Problems ECP and VCP ask for the largest set of directed edge (node) disjoint cycles in a given graph $G(V, E)$. We denote the maximum

number of edge (vertex) disjoint cycles in a graph G by $|ECP(G)|$ ($|VCP(G)|$). We can also define fractional versions of these problems as follows. Let C be a set that includes all cycles in the graph and let $\psi, C \rightarrow R$ be a function that maps each cycle $c \in C$ to a real number. Our goal is to find a function ψ that maximizes $\sum_{c \in C} \psi(c)$ subject to the following constraints:

- For each $v \in V$, it holds that $\sum_{v \in c, c \in C} \psi(c) \leq 1$ (for node-disjoint case);
- For each $e \in E$, it holds that $\sum_{v \in c, c \in C} \psi(c) \leq 1$ (for edge-disjoint case).

We denote by $|ECP^f(G)|$ ($|VCP^f(G)|$) the optimal value of the fractional edge-disjoint (node-disjoint) cycle packing problem.

C. Multiple Unicast Case

In this section we first present an approximation algorithm for the scalar version of the *Complementary Index Coding (CIC)* problem. Then, we present an algorithm for the vector version of this problem.

1. An Approximation Algorithm for Finding Scalar-Linear Solution

The main idea of our algorithm is to find a vertex disjoint cycle packing in the dependency graph. Note that for each vertex-disjoint cycle in the dependency graph we can save at least one transmission. To see this, consider the example depicted in Figure 30. In this example, we have a cycle that involves five clients, such that client c_i requires packet p_i. For $i = 2, \ldots, 5$ it holds that the client c_i has the packet required by client c_{i-1}. It is easy to verify that all clients can be satisfied by four transmissions: $p_1 + p_2$, $p_2 + p_3$, $p_3 + p_4$, and $p_4 + p_5$ (all operations are over $GF(2)$). Indeed, the client c_2 will be satisfied by the transmission $p_1 + p_2$, the client c_3 will be satisfied by

Algorithm *sCIC* ()
1 From the given instance of the *CIC* problem, construct the *Dependency Graph* $G(V, E)$;
2 From the given dependency graph construct the *Vertex Split Graph* $G'(V', E')$
3 $C' = \emptyset$
4 **While** there exists a directed cycle in $G'(V', E')$ **do**:
5 Find a cycle c' of minimum length
6 Add c' to C'
7 Delete all the edges of c' from $G'(V', E')$
8 **For** each cycle $c' \in C'$ **do**:
9 Identify the corresponding cycle c in $G(V, E)$;
10 Transmit a set of $|c| - 1$ transmissions that satisfy all clients in c
11 For each $v_i \in G(V, E)$ not included in any cycle c, transmit the packet required by the corresponding client c_i

Fig. 29. Algorithm *sCIC*

transmission $p_2 + p_3$, and so on. The client c_1 will add all the transmissions to obtain $p_1 + p_5$, which will allow it to decode packet p_2.

Thus, a vertex-disjoint cycle packing of size k will allow to find a solution that saves k transmissions. We summarize our result by the following lemma.

Lemma 15 *Let α be a set of vertex disjoint cycles in the dependency graph $G(V, E)$. Then, it is possible to construct a feasible solution to the* CIC *problem that saves at least $|\alpha|$ transmissions, where $|\alpha|$ is the size of α.*

For convenience, we convert the problem of finding vertex-disjoint cycle packing in the dependency graph $G(V, E)$ to the problem of finding edge-disjoint cycle packing in the vertex split graph $G'(V', E')$ of $G(V, E)$. As mentioned above, the size of the edge-disjoint cycle packing in graph $G'(V', E')$ is equal to the size of vertex-disjoint cycle packing in $G(V, E)$.

Our algorithm, referred to as Algorithm *sCIC*, performs the following steps. First, the algorithm constructs the dependency graph $G(V, E)$ for the problem at

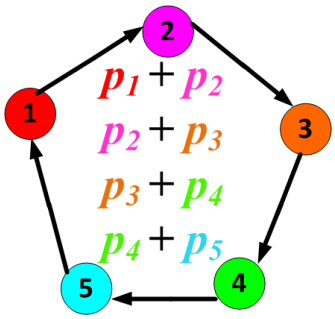

Fig. 30. The *dependency graph* of a cycle and corresponding optimal set of transmissions.

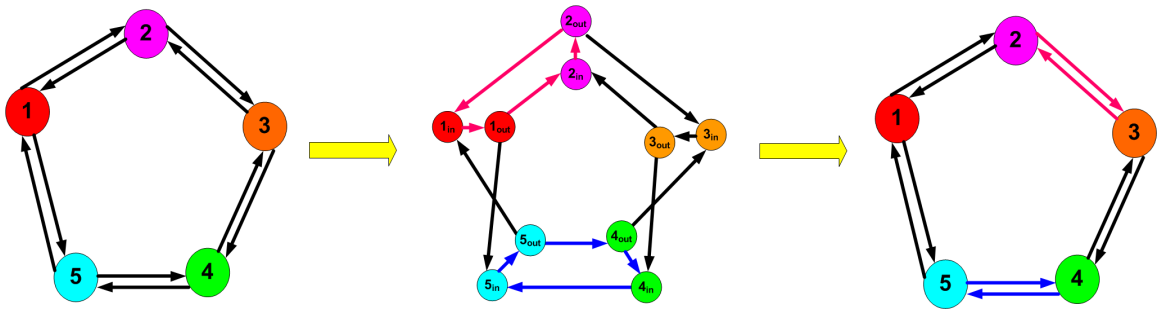

Fig. 31. A sample execution of the Algorithm *sCIC* on the instance of the *CIC* problem shown in Figure 22.

hand. Next, the vertex split graph $G'(V', E')$ of $G(V, E)$ is constructed. Finally, we apply the approximation algorithm due to Krivelevich et al. [41] to find an approximate cycle packing in $G'(V', E')$. Next, we identify the set of vertex-disjoint cycles in $G(V, E)$ that correspond to edge-disjoint cycles in $G(V, E)$. Finally, for each cycle in the dependency graph we identify the set of encoding vectors such that one transmissions are saved per cycle. The formal description of Algorithm *sCIC* is presented in Figure 29. Figure 31 shows a sample execution of the Algorithm *sCIC* for the instance shown in Figure 22.

We proceed to analyze the correctness of Algorithm *sCIC*.

Lemma 16 *Algorithm* sCIC *finds a scalar-linear solution to the Complementary Index Coding problem with approximation ratio of* $\Omega(\sqrt{n} \cdot \log n \cdot \log\log n)$.

Proof: By Proposition 13, $OPT_{CIC} \leq |FVS(G)|$. As discussed above, $|FVS(G)| = |FES(G')|$, where G' is the vertex split graph of G, hence $OPT_{CIC} \leq |FES(G')|$. By [42], the integrality gap of the Feedback Edge Set problem is bounded by $\log n \cdot \log\log n$, hence $|FES(G')| \leq |FES^f(G')| \log n \cdot \log\log n$. Hence, $OPT_{CIC} \leq |FES^f(G')| \log n \cdot \log\log n$. The algorithm due to [41] yields an edge disjoint cycle cover C', whose size is at least $|FES^f(G')|/\sqrt{n}$. Since for each cycle we save a transmission, the total number of saved transmissions is at least

$$\frac{|FES^f(G')|}{\sqrt{n}} \geq \frac{|FES(G')|}{\sqrt{n} \cdot \log n \cdot \log\log n} \geq \frac{OPT_{CIC}}{\sqrt{n} \cdot \log n \cdot \log\log n}.$$

■

2. An Approximation Algorithm for Finding Vector-Linear Solution

In this section we present an algorithm, referred to as Algorithm *vCIC*, for finding vector-linear solutions for the *CIC* problem. The algorithm achieves the approximation ratio of $\Omega(\log n \cdot \log\log n)$. As mentioned in the model, with a vector-linear solution, each packet can be divided into several smaller-size subpackets and the server can transmit linear combinations of these subpackets.

The algorithm includes the following steps. Given an instance of the *CIC* problem, we first construct the dependency graph $G(V, E)$, and the corresponding Vertex Split Graph $G'(V', E')$. Then, we apply the algorithm due to Yuster and Nutov [40] to find the fractional cycle packing in $G'(V', E')$, which, in turn, yields a fractional vertex-disjoint cycle packing $\psi : C \to R$ in $G(V, E)$. Then, we find the minimum

integer number k that satisfies that $k\psi(c)$ is an integer for any $c \in C$. Such number exists because for each $c \in C$, $\psi(c)$ is a rational number. Next, we divide each packet $p_i \in P$ into k smaller size subpackets $p_i^1, \ldots, p_i^k$.[†]

Next, we create an auxiliary dependency graph $\hat{G}(\hat{V}, \hat{E})$. This graph is constructed similarly to the regular dependency graph, but for subpackets, instead of the original packets. Specifically, graph $\hat{G}(\hat{V}, \hat{E})$ is defined as follows:

- For each subpacket p_i^j of a packet $p_i \in P$ there is a corresponding vertex v_i^j in $\hat{V}$
- There is a directed edge from v_i^j and v_l^h if and only if it holds that $p_i \in H(c_l)$, c_l is a client requesting packet p_l.

Graph $\hat{G}(\hat{V}, \hat{E})$ has the following property. For each fractional cycle packing ψ of graph $G(V, E)$ of size α, there exists a set of vertex-disjoint cycles $\hat{C}$ in $\hat{G}(\hat{V}, \hat{E})$ of size αk. The integer cycle packing $\hat{C}$ in $\hat{G}(\hat{V}, \hat{E})$ can be identified through the following procedure. For each cycle $c \in C$ for which $\psi(c) > 0$ we can identify $k \cdot \psi(c)$ vertex-disjoint cycles in $\hat{G}$ such that for node $v_i \in C$, each of the corresponding cycles use one of the nodes in $\{v_i^1, \ldots, v_i^k\}$. We then remove $k \cdot \psi(c)$ vertex-disjoint cycles from $\hat{G}$ and repeat the procedure for the next cycle in C.

Now, for each cycle $\hat{c} \in \hat{C}$ we generate $|\hat{c}| - 1$ linear combinations that satisfy the demands for packets that correspond to vertices in $\hat{c}$. Each such cycle will *save* one subpacket, so in total αk subpackets will be saved. This corresponds to saving α original packets.

[†]This can be done assuming that each packet is sufficiently large and the size of each packet in P is a multiple of k. Thus, in practice, a fractional solution can be suitable for large packets. This assumption is standard for vector-linear network coding schemes.

A sample execution of the Algorithm *vCIC* on the instance of the *CIC* problem is shown in Figure 32. This Figure shows the auxiliary graph $\hat{G}(\hat{V}, \hat{E})$ and the corresponding 5 cycles packed in the *auxiliary dependency graph*. Therefore the corresponding transmissions are: $p_1^1 + p_2^1$, $p_2^2 + p_3^1$, $p_3^2 + p_4^1$, $p_4^2 + p_5^1$, and $p_5^2 + p_1^2$.

We proceed to analyze the correctness of the Algorithm *vCIC*.

Lemma 17 *Algorithm* vCIC *finds a vector-linear solution to the* Complementary Index Coding *problem with approximation ratio of* $\Omega(\log n \cdot \log\log n)$.

Proof: Let OPT_{CIC}^f be the value of the optimal vector-linear solution to the *CIC* problem. It is easy to verify that, similar to the integer case, $OPT_{CIC}^f \leq |FVS(G)|$. This follow from the fact that the optimal vector-linear solution to IC is greater or equal than the size of $MAIS(G)$.

Since $|FVS(G)| = |FES(G')|$, it holds that $OPT_{CIC}^f \leq |FES(G')|$. By [42], the integrality gap of the Feedback Edge Set problem is bounded by $\log n \cdot \log\log n$, hence $|FES(G')| \leq |FES^f(G')| \log n \cdot \log\log n$. Hence, $OPT_{CIC}^f \leq |FES^f(G')| \log n \cdot \log\log n$. As discussed above, our solution saves $|FES^f(G')|k$ subpackets, which is equivalent to saving $|FES^f(G')|$ packets for the *CIC* problem. Thus, the total number of saved transmissions is at least $\frac{OPT_{CIC}^f}{\log n \cdot \log\log n}$ and the lemma follows.

■

D. Multiple Multicast Case

In this section we allow multiple clients to require the same packet i.e., $m \geq n$, and show that in this case *Complementary Index Coding (CIS)* problem is not only NP-hard but is also hard to approximate.

We prove the results by reducing the *Independent Set (IS)* problem (where the

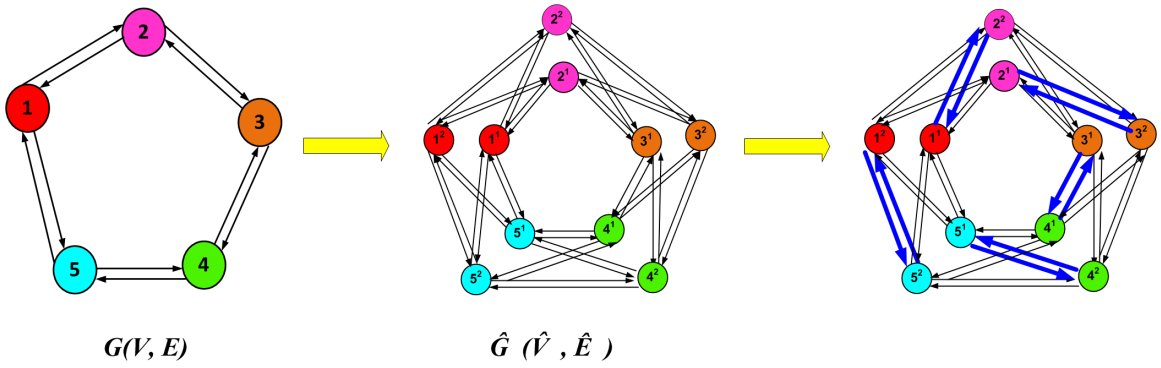

Fig. 32. A sample execution of the Algorithm *vCIC* on the instance of the *CIC* problem shown in Figure 22.

objective is to find the Independent Set of maximum cardinality) into Problem *CIC*. We denote the value of the optimal solution of the *IS* problem by OPT_{IS}. The instance of the *IS* problem is given by a graph $G(V, E)$.

Given an instance of the IS problem we define an instance of the *CIC* problem as following:

- For each vertex $v \in V$, we define a packet p_v, and for each edge $e = (u, v) \in E$, we define packet p_e, i.e, we define a total of $|V| + |E|$ packets.
- For each edge $e = (u, v) \in E$ we define the following three clients $c_{e_1}, \cdots, c_{e_3}$ such that
  1. Client c_{e_1} requires packet p_u and has packet p_e;
  2. Client c_{e_2} requires packet p_v and has packet p_e;
  3. Client c_{e_3} requires packet p_e and has packets p_u and p_v.

  i.e., we define a total of $3 \cdot E$ clients.

This construction is similar to that used in our previous work [35].

Lemma 18 $OPT_{IS} = OPT_{\text{CIC}}$.

Proof: We proceed to prove that $OPT_{IS} = OPT_{CIC}$. Let OPT_{VC} be value of the minimum Vertex Cover in $G(V, E)$. In [35] it was shown that

$$OPT_{VC} + |E| = OPT_{IC} \tag{5.1}$$

Since a set of vertices in G that do not belong to a vertex cover constitute an independent set, it holds that

$$OPT_{VC} + OPT_{IS} = |V| \tag{5.2}$$

Also, since the total number of packet in our instance of Problem CIC is equal to $|V| + |E|$, it holds that

$$OPT_{CIC} = |V| + |E| - OPT_{IC} \tag{5.3}$$

By combining Equations 5.1-5.3 we get $OPT_{IS} = OPT_{CIC}$. ■

By Lemma 18, the optimal value of Problem *CIC* is equal to the maximum size of the independent set in $G(V, E)$. Thus, our reduction implies that the Complementary Index Coding is NP-hard. Furthermore, since it is NP-hard to approximate the independent set problem within a ratio of $n^{1-\epsilon}$ [43] the same holds for Problem *CIC*.

We conclude our discussion by the following theorem.

Theorem 19 *The Complementary Index Coding is NP-hard, and it is NP-hard to approximate within a ratio of* $n^{1-\epsilon}$ *for any constant* $\epsilon > 0$.

E. Performance Evaluation

In this section we evaluate the performance of the proposed scalar-linear solution for the *Complementary Index Coding* problem in unicast settings. More specifically,

we evaluate the performance of the algorithm *sCIC* described in Figure 29, and the algorithm *Imp-sCIC* described in Figure 33.

The experimental setup is as follows. We consider n clients, where each client c_i requires packet p_i. The *has* set $H(c_i)$ for each client c_i is chosen randomly. More specifically, for a given cardinality of $H(c_i)$ say ℓ, for each client c_i, we randomly choose ℓ packets for $H(c_i)$ out of n packets $\{p_1, \cdots, p_n\}$. Throughout this section each value in the simulation plots represents an average over 100 experiments.

The improved algorithm for the scalar version of the *CIC* problem, i.e., the Algorithm *Imp-sCIC* consists of two phases. In *Phase 1* we use the *Color Saving* based heuristic given in Section IV.B.3, i.e., in the Dependency Graph $G(V, E)$ we first find as many *3-cliques* as possible, and then find all possible *2-cliques*. After finding each clique we delete the corresponding vertices. Note that the cliques involve only bi-directed edges. A bi-directed edge (v_i, v_j) in $G(V, E)$ shows that for clients c_i and c_j, $w_j \in H(c_i)$, and $w_i \in H(c_j)$. Note that a set of clients forming a clique can be served in a single transmission which is equal to the summation of their *wants* set. Then in *Phase 2*, we invoke the Algorithm *sCIC* on the graph left after phase 1 (which does not have any bi-directed edges left). The intuition lies in the fact that the number of the transmissions saved for a clique of size k is $k-1$ whereas only one transmission can be saved for each cycle. Therefore packing as many cliques as possible will increase the overall number of transmissions saved. The formal description of the Algorithm *Imp-sCIC* is given in Figure 33.

Figure 34 shows the comparison of the algorithms *sCIC* and *Imp-sCIC* for the average percentage of the transmissions saved versus the cardinality of the *has* set for 100 clients. Figure shows that with the increased cardinality of the *has* set the number of transmissions saved increases. This can be intuitively explained as follows:

Algorithm *Imp-sCIC* ()
1 **Phase 1:**
2 As long as there exist any *3-clique* in the *Dependency Graph* $G(V, E)$,
3 Find *3-clique* $\{v_i, v_j, v_k\}$ and delete the corresponding vertices i.e., $V := V \setminus \{v_i, v_j, v_k\}$
4 The server transmits the packet $p_i + p_j + p_k$ for the *3-clique* $\{v_i, v_j, v_k\}$
5 Find all the *2-cliques* in $G(V, E)$ using maximum matching
6 For each *2-clique* $\{v_i, v_j\}$ delete the corresponding vertices i.e., $V := V \setminus \{v_i, v_j\}$
7 The server transmits the packet $p_i + p_j$ for the *2-clique* $\{v_i, v_j\}$
8 **Phase 2:**
9 Apply the algorithm *sCIC* on $G(V, E)$.

Fig. 33. Algorithm *Imp-sCIC*

increased cardinality of the *has* set indicates more side information available to the clients, so the number of edges in the Dependency Graph increase, that results in more number of cliques and cycles and hence more savings. Figure also shows that the Algorithm *Imp-sCIC* results in more saved transmissions as compared to the Algorithm *sCIC*. Furthermore, for each point in the plot a 90% confidence interval is also shown.

Figure 35 shows the average percentage of the transmissions saved using the Algorithm *Imp-sCIC* by varying the number of clients. In this set of experiments the cardinality of the *has* set is selected randomly. The increase in the number of transmissions saved is due to the fact that the number of edges in the graph increases with the increase in the number of the clients which results in larger number of cliques and cycles. Furthermore, a 90% confidence interval is also shown.

Figure 36 shows the result of two sets of experiments for the average percentage of the transmissions saved versus number of the clients using the Algorithm *Imp-sCIC*.

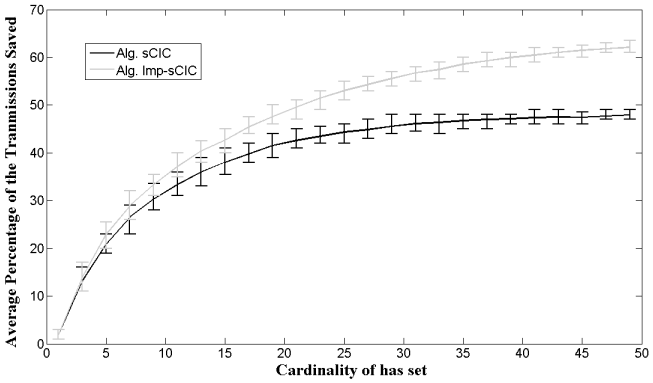

Fig. 34. Average percentage savings in the number of transmission using the algorithms *sCIC*, and *Imp-sCIC*. Plot shows the average percentage of savings versus the cardinality of the *has* set for 100 clients. Plot also shows the 90% confidence interval for both settings.

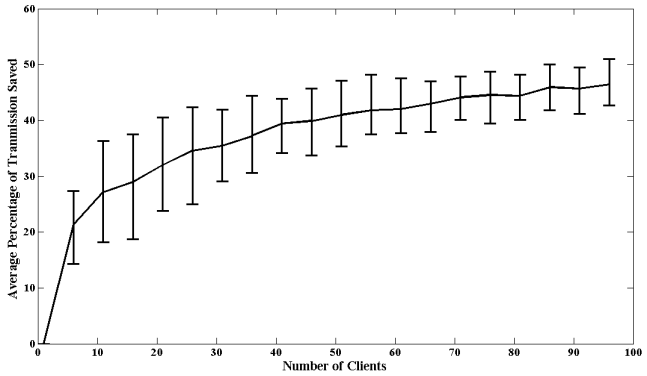

Fig. 35. Average percentage of the number of the transmissions saved versus the number of clients using the Algorithm *Imp-sCIC*. Plot shows the average percentage of the savings versus the number of clients for random cardinality of the *has* set for each client. Plot also shows the 90% confidence interval.

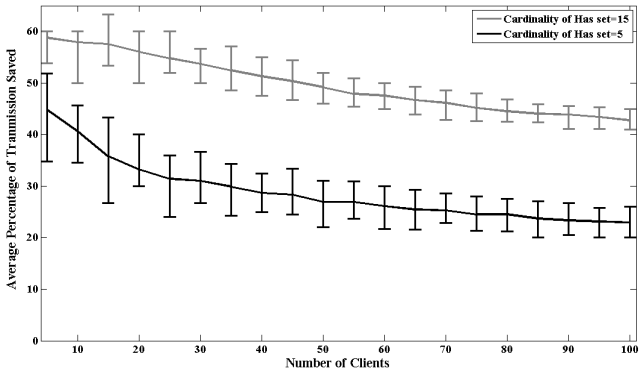

Fig. 36. Percentage of the transmissions saved using the Algorithm *Imp-sCIC*. Plot shows the average percentage of savings versus the number of clients with cardinality of the *has* set fixed to the value 5 and 15. Plot also shows the 90% confidence interval for both settings.

In the first set we fixed the cardinality of *has* set to 5, while in the second set we fixed the cardinality of *has* set to 15. As mentioned before, the *has* sets were chosen randomly, and average was taken over 100 experiments. Plot also shows confidence interval of 90%.

Figure 37 shows the breakup of both phases of the Algorithm *Imp-sCIC* for the average percentage of the transmissions saved versus the number of clients, for a fixed cardinality of *has* set. Figure shows that when the cardinality of the *has* is comparable to the number of clients then *Phase 1* of the Algorithm *Imp-sCIC* contributes more towards saving of transmissions, but as the cardinality of the *has* becomes smaller compared to the number of clients the advantage of the *Phase 1* keeps on decreasing and *Phase 2* starts contributing more towards saving of transmissions. The reason behind this observation is that when the cardinality of the *has* set is comparable to the number of clients then the Dependency Graph is more dense and there exits more

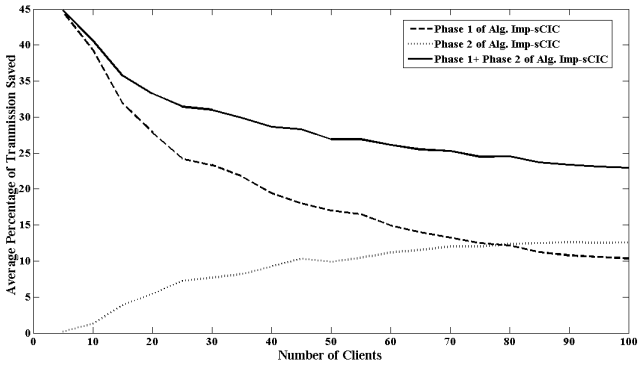

Fig. 37. Breakup of the percentage of the transmissions saved using the Algorithm *Imp-sCIC*. Plot shows the average percentage of savings versus the number of clients with the cardinality of *has* set equals to 5.

bi-directed edges and so the Algorithm *Imp-sCIC* can find more cliques in the *Phase 1*. But when the cardinality of the *has* set is small as compared to the number of clients the Dependency Graph is sparse and it contains lesser number of cliques, so the Algorithm *Imp-sCIC* can pack more cycles in the *Phase 2*, and hence the *Phase 2* saves more transmissions as compared to the *Phase 1*.

F. Conclusion

In this chapter, we focused on the complementary problem of the *Index Coding* problem (referred to as the *Complementary Index Coding* (CIC) problem). The goal of the CIC problem is to maximize the number of *saved transmissions*, i.e., the number of transmissions that are saved by using the Index Coding technique compared to the traditional solution that does not involve coding.

We considered two scenarios for the CIC problem, i.e., the *multiple unicast* scenario and the *multiple multicast* scenario. In the multiple unicast scenario, each

packet is requested by a single client; whereas in the multiple multicast scenario, each packet can be requested by more than one client. We also considered *scalar* and *vector* linear solutions for the problem at hand. In the scalar linear solution, the packets cannot be split. In a vector-linear solution, each packet can be split into a smaller sub-packets, such that sub-packet originated from different packets can be combined together. For the multiple unicast scenario, we presented approximation algorithms for finding scalar and vector-linear solutions. The approximation ratios of the scalar and vector-linear solutions are $\Omega(\sqrt{n} \cdot \log n \cdot \log \log n)$ and $\Omega(\log n \cdot \log \log n)$, respectively. For the multiple muticast scenario, we showed that finding an approximate solution within ratio $n^{1-\epsilon}$ is NP-hard.

CHAPTER VI

SPARSE SOLUTIONS TO INDEX CODING PROBLEM*

In this chapter, we focus on finding *sparse* solutions to the Index Coding problem. In a sparse solution each transmitted packet is a linear combination of at most two original packets. We focus both on *scalar* and *fractional* versions of the problem. For the scalar case, we present a polynomial time algorithm that achieves an approximation ratio of $2 - \frac{1}{\sqrt{n}}$. For the fractional case, we present a polynomial time algorithm that identifies the optimal solution to the problem. Our simulation studies demonstrate that our algorithms achieve good performance in practical scenarios.

A. Introduction

The previous works on the Index Coding problem considered the general setup where the server can encode as many packets as necessary. In this chapter, we are focusing on a practically important special case, in which a server can mix at most *two packets* in any single transmission. We refer to this problem as the *Sparse Index Coding (SIC)* problem. Consider an instance of the *IC* problem shown in Figure 38 in which a server needs to deliver three packets p_1, p_2, p_3 to three clients. Each client *wants* one packet and *has* other two packets. An optimal solution to the *IC* problem can satisfy all clients by transmitting a single packet $p_1 + p_2 + p_3$. Note that in the case of the *SIC* problem where the server is restricted to encode at most two packets, the optimal scalar-linear solution can save 1 transmission by transmitting two packets $p_1 + p_2$,

*Parts of this chapter are reprinted with permission from "Finding Sparse Solutions for the Index Coding Problem" by M. A. R. Chaudhry, Z.asad, A. Sprintson, and M. Langberg to appear in the proceedings of the 2011 IEEE Global Communications Conference (GLOBECOM), Houston, U.S.A., 2011.

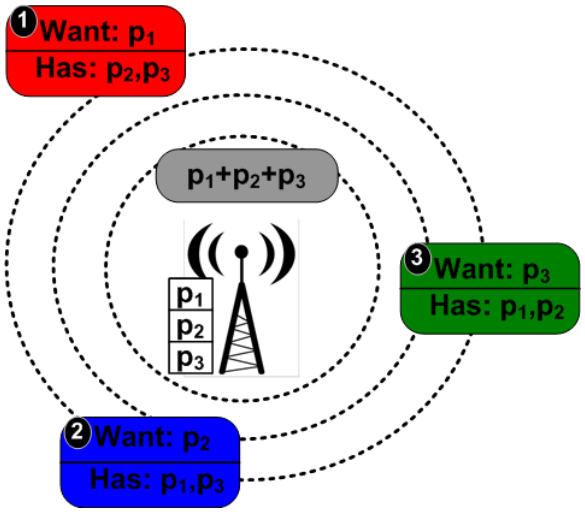

Fig. 38. An instance of the Index Coding.

and $p_2 + p_3$. However in the case of the vector-linear solution to the *SIC* problem the optimal can save 1.5 transmissions by transmitting the packets $p_1^1 + p_2^1$, $p_2^2 + p_3^1$, and $p_3^2 + p_1^2$, where each packet p_i is divided into two half packets p_i^1, and p_i^2. With sparse network coding, the encoders and decoders can be implemented very efficiently which makes it attractive for practical applications. Furthermore, sparse Index Coding can be implemented over a small field ($GF(2)$), which allows to significantly reduce the size of the packet headers and associated overhead.

In Chapter V we presented the concept of the *Complementary Index Coding (CIC)* problem. In the *CIC* problem, instead of minimizing the number μ of transmissions, the goal is to maximize the number of "saved" transmissions, i.e., $n - \mu$, where n is the number of packets that need to be delivered to the clients. The *CIC* problem seeks to maximize the benefit obtained by employing the coding technique, e.g., for the instance shown in Figure 38, the number of transmissions saved is 2. Note that, if OPT_{IC} is the optimum of the Index Coding problem and OPT_{CIC}

is the optimum of the Complementary Index Coding problem, then it holds that $|OPT_{CIC}| = n - |OPT_{IC}|$. This implies that finding the scalar-linear solution for the *CIC* problem is also NP-hard. However, as shown in the Chapter V the *CIC* problem can be successfully approximated in many cases of practical importance as compared to the Index Coding problem which has been proven to be hard to approximate. More specifically, the Chapter V presents algorithms with the approximation ratios of $\Omega(\sqrt{n} \cdot \log n \cdot \log \log n)$ and $\Omega(\log n \cdot \log \log n)$, for the scalar and vector linear solutions for the *CIC* problem, respectively.

In this chapter we investigate the *Sparse Index Coding* problem, and consider *scalar* and *vector* linear solutions. In the scalar-linear solution, the packets cannot be split. In a vector-linear solution, each packet can be split into a smaller sub-packets, such that the sub-packets originated from different packets can be combined together. First, we establish a connection between the sparse scalar Index Coding problem and the problem of finding the maximum number of vertex-disjoint cycles (i.e., the cycle packing problem). Thus connection implies that the sparse Index Coding problem is NP-hard. Second, for we present an algorithm that achieves an approximation ratio of $2 - \frac{1}{\sqrt{n}}$ for the scalar-linear case, i.e., with our algorithm the number of transmissions is at most $2 - \frac{1}{\sqrt{n}}$ times the optimum. We note that for the sparse Index Coding, the approximation ratio of 2 can be achieved by using the standard approach (which does not include network coding). However, our solution allows to avoid ("save") at least $\frac{1}{\sqrt{n}}$ transmissions compared to the standard solution. Next, we present an algorithm that identifies an optimal vector-linear solution in polynomial time. Finally, we present an experimental study showing the advantage of the proposed algorithms.

B. Finding Efficient Scalar-linear Solution

In this section, we focus on scalar-linear solutions of the SIC problem. The key idea is to establish a connection between Problem SIC and the problem Problem *Vertex Cycle Packing* in the corresponding *dependency* graph $G(V, E)$ given in Section V.B.

Problem *Vertex Cycle Packing* (VCP) asks for the largest set of vertex-disjoint cycles in the graph $G(V, E)$. We denote the optimal solution to Problem VCP by OPT_{VCP}.

The main idea is to show that for each vertex-disjoint cycle in the dependency graph we can save at least one transmission. To see this, consider the example depicted in Figure 30. In this example, we have a cycle that involves five clients, such that client c_i requires packet p_i. For $i = 2, \ldots, 5$ it holds that the client c_i has the packet required by client c_{i-1}. It is easy to verify that all clients can be satisfied by four transmissions: $p_1 + p_2$, $p_2 + p_3$, $p_3 + p_4$, and $p_4 + p_5$. Indeed, the client c_2 will be satisfied by the transmission $p_1 + p_2$, the client c_3 will be satisfied by transmission $p_2 + p_3$, and so on. The client c_1 will add all the transmissions to obtain $p_1 + p_5$, which will allow it to decode packet p_1.

Our algorithm, referred to as Algorithm *sSIC*, performs the following steps. First, the algorithm constructs the dependency graph $G(V, E)$ for the problem at hand. Next, the *vertex split* graph $G'(V', E')$ of $G(V, E)$ is constructed. The vertex-split graph $G'(V', E')$ is formed from $G(V, E)$ by substituting each vertex $v_i \in V$ by two vertices v_i' and v_i'' and an edge (v_i', v_i'') that connects v_i' and v_i''; and by substituting each edge $(v_i, v_j) \in E$ by an edge (v_i'', v_j'). Finally, we apply the approximation algorithm due to Krivelevich et al. [41] to find an approximate cycle packing in $G'(V', E')$. Next, we identify the set of vertex-disjoint cycles in $G'(V', E')$ that correspond to edge-disjoint cycles in $G(V, E)$. Finally, for each cycle in the dependency graph we

Algorithm *sSIC* ()
1 From the given instance of the *IC* problem, construct the *dependency graph* $G(V, E)$;
2 From the given dependency graph construct the *Vertex Split Graph* $G'(V', E')$
3 $C' = \emptyset$
4 **While** there exists a directed cycle in $G'(V', E')$ **do**:
5 Find a cycle c' of minimum length
6 Add c' to C'
7 Delete all the edges of c' from $G'(V', E')$
8 **For** each cycle $c' \in C'$ **do**:
9 Identify the corresponding cycle c in $G(V, E)$;
10 Transmit a set of $|c| - 1$ transmissions that satisfy all clients in c
11 For each $v_i \in G(V, E)$ not included in any cycle c, transmit the packet required by the corresponding client c_i

Fig. 39. Algorithm *sSIC*

identify the set of encoding vectors such that one transmissions is saved per cycle. Algorithm *sSIC* has a running time of $O(n^3)$.

The formal description of Algorithm *sSIC* is presented in Figure 39.

We proceed to analyze the correctness of Algorithm *sSIC*. In the following two lemmas we prove that $n - OPT^s = OPT_{VCP}$.

Lemma 20 $n - OPT^s \geq OPT_{VCP}$.

Proof: Let $S = \{s_1, s_2, \ldots, s_m\}$ be a packing of vertex-disjoint cycles in $G(V, E)$. Then, we construct an solution to Problem SIC that includes, for each cycle $s_i = \{v_{i_1}, v_{i_2}, \ldots, v_{i_l}, v_{i_1}\}$ packets $p_{i_j} + p_{i_{j+1}}$ for $j = 1, \ldots, l-1$, where l is the size of the cycle. It is easy to verify that the total number of transmitted packets is equal to $n - m$, i.e., for each cycle, one transmission is "saved." Also, it is easy to see that this is a valid solution to Problem CIS. Indeed, each client c_{i_j}, $2 \leq j \leq l$ can recover its required packets w_{i_j} directly from transmitted packet $p_{i_{j-1}} + p_{i_j}$. We note that

$$\sum_{j=1}^{l-1} p_{i_j} + p_{i_{j+1}} = p_{i_l} + p_{i_1}.$$

Thus, client c_{i_1} can also recover the packet it requires. ■

Next, we show that $OPT_{VCP} \geq n - OPT^s$.

Lemma 21 $n - OPT^s \leq OPT_{VCP}$.

Proof: Let $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_{OPT^s}\}$ be an optimal solution to Problem SIC. Note that each $\gamma_i \in \Gamma$ is a combination of at most two packets in P, these packets are referred to as the *support* of γ_i. First, we define a set $P_1 \subseteq P$ that includes all packets $p_i \in P$ for which it holds that p_i belong to the span of Γ. We define $P_2 = P \setminus P_1$. We say that two packets $p_i \in P_2$ and $p_j \in P_2$ are *connected* if a linear combination of p_i and p_j belongs to the span of Γ. Note that the connectivity is a transitive property, hence P_2 can be divided to equivalence classes $P_2^1, P_2^2, \ldots$, such that each equivalence class includes connected packets. Note that the number of equivalence classes is equal to $n - OPT^s$.

Let $V_2^1, V_2^2, \ldots$ be subsets of vertices of V that correspond to equivalence classes $P_2^1, P_2^2, \ldots$. We show that for each V_2^i it holds that the subgraph of G induced by V_2^i contains at least one cycle. Since the subsets $V_2^1, V_2^2, \ldots$ are disjoint, this will be sufficient to show that $G(V, E)$ contains at least $n - OPT^s$ disjoint cycles.

Let V_2^i be a subset that includes two or more nodes and let G^i be a subgraph of G induced by V_2^i. We show that the in-degree of each node $v_j \in G^i$ is at least one. Indeed, let v_j be a node in V_2^i and let c_j be the client that corresponds to v_j. Let $\hat{\gamma}$ be a vector in span of Γ used by c_j to decode packet w_j in its wants set. It is easy to see that $\hat{\gamma}$ includes at least one packet, say v_l in P_2^i. Then, there exists an edge (v_l, v_j) in G^i and the lemma follows. ■

For example, consider an instance of the *IC* problem as shown in Figure 22. The

optimal solution to the Problem *SIC* is given by: $\Gamma = \{p1 + p2, p3 + p4, p5\}$. In this example: $P_1 = \{p_5\}$, and $P2 = \{p_1, p_2, p_3, p_4\}$. The corresponding equivalence classes are as follows: $P_2^1 = \{p_1, p_2\}$ and $P_2^2 = \{p_3, p_4\}$, with $V_2^1 = \{v_1, v_2\}$ and $V_2^2 = \{v_3, v_4\}$ respectively. The subgraph corresponding to V_2^1 consists of two arcs (v_1, v_2) and (v_2, v_1), and hence corresponds to a cycle between v_1 and v_2. Similarly the subgraph corresponding to V_2^2 contains a cycle between v_3 and v_4.

Lemma 22 Sparse Index Coding (SIC) problem *is an NP-Complete problem. Furthermore, it is quasi-NP-hard to approximate the number of transmissions "saved ", i.e., $n - OPT^s$, within a factor of $O(\log^{1-\varepsilon} n)$ for any constant $\varepsilon > 0$.*

Proof: By combining lemmas 20 and 21 we get $n - OPT^s = OPT_{VCP}$. Then, by using the inapproximability result of the *VCP* problem [41], the lemma follows.

■

We conclude with the following theorem:

Theorem 23 *Algorithm* sSIC *finds a scalar-linear solution to the Sparse Index Coding problem with approximation ratio of $2 - \frac{1}{\sqrt{n}}$. The algorithm also allows to "save" at least a factor of $\frac{1}{\sqrt{n}}$ times the optimum saving.*

Proof: Let OPT^s be the optimal solution for Problem SIC. By lemmas 20 and 21, the maximum number of vertex-disjoint cycles that can be packed in $G(V, E)$ is $OPT_{VCP} = n - OPT^s$. Then, by using the approximation algorithm due to Krivelevich et al. [41] we can identify at at least $\frac{OPT_{VCP}}{\sqrt{n}}$ cycles. Thus, our algorithm requires at most

$$n - \frac{OPT_{VCP}}{\sqrt{n}} = n - \frac{n - OPT^s}{\sqrt{n}} \tag{6.1}$$

transmissions. This implies that the algorithm achieves an approximation ratio of

$$\frac{1}{\sqrt{n}} + \frac{n - \sqrt{n}}{OPT^s}. \tag{6.2}$$

Since $OPT^s \geq \frac{n}{2}$, Equation (6.2) implies that the approximation ratio is bounded by $2 - \frac{1}{\sqrt{n}}$.

By Equation (6.1), the algorithm "saves" at least $\frac{n - OPT^s}{\sqrt{n}}$ transmissions compared the standard solution that does not use coding. Since the optimal solution to Problem SIC saves $n - OPT^s$ transmissions, that algorithm allows to save at least a factor of $\frac{1}{\sqrt{n}}$ times the optimum saving. ■

C. Finding Efficient Vector-linear Solution

In this section, we present an algorithm, referred to as Algorithm *vSIC*, that finds an optimum vector-linear solution to Problem SIC. The algorithm exploits the connection between Problem SIC and the problem of finding an optimal fractional solution for the cycle packing problem, defined as follows. Let C be a set that includes all cycles in the graph $G(V, E)$ and let $\psi, C \rightarrow R$ be a function that maps each cycle $c \in C$ to a real number. Our goal is to find a function ψ that maximizes $\sum_{c \in C} \psi(c)$ such that for each $v \in V$, it holds that $\sum_{c \in C} \psi(c) \leq 1$. We denote by OPT^f_{VCP} the optimum fractional solution to the vertex-disjoint cycle packing.

The algorithm includes the following steps. Given an instance of Problem *IC*, we first construct the dependency graph $G(V, E)$. Then, we apply the algorithm due to Yuster and Nutov [40] to find an optimal vertex-disjoint cycle packing $\psi : C \rightarrow R$ in $G(V, E)$.[†] Then, we find the minimum integer number k for which it holds that

[†]The algorithm due to [40] finds an optimal fractional edge-dijoint cycle packing,

$k\psi(c)$ is an integer for any $c \in C$. Such number exists because for each $c \in C$, $\psi(c)$ is a rational number. Next, we divide each packet $p_i \in P$ into k smaller size subpackets $p_i^1, \ldots, p_i^k$.

Next, we create a fractional dependency graph $\hat{G}(\hat{V}, \hat{E})$. This graph is constructed similarly to the dependency graph for the scalar case, with the difference that the nodes in $\hat{V}$ correspond to subpackets, and not to the original packets. Specifically, graph $\hat{G}(\hat{V}, \hat{E})$ is defined as follows:

- For each subpacket p_i^j of a packet $p_i \in P$ there is a corresponding vertex v_i^j in $\hat{V}$
- There is a directed edge from v_i^j and v_l^h if and only if it holds that $p_i \in H(c_l)$, where c_i are c_l are clients requesting packets p_i and p_l, respectively.

Graph $\hat{G}(\hat{V}, \hat{E})$ has the following property. For each fractional cycle packing ψ of graph $G(V, E)$ of size α, exists a set of vertex-disjoint cycles $\hat{C}$ in $\hat{G}(\hat{V}, \hat{E})$ of size αk. Given a fractional cycle packing ψ in $G(V, E)$, the integer cycle packing $\hat{C}$ in $\hat{G}(\hat{V}, \hat{E})$ can be identified through the following procedure. For each cycle $c \in C$ for which it holds that $\psi(c) > 0$ we can identify $k \cdot \psi(c)$ vertex-disjoint cycles in $\hat{G}$ such that for each node $v_i \in C$, each of the corresponding cycles use one of the nodes in $\{v_i^1, \ldots, v_i^k\}$. We then remove $k \cdot \psi(c)$ vertex-disjoint cycles from $\hat{G}$ and repeat the procedure for the next cycle in C.

Now, for each cycle $\hat{c} \in \hat{C}$ we generate $|\hat{c}|-1$ linear combinations of the subpackets $\{p_i^j\}$ that correspond to vertices in $\hat{c}$. Each such cycle will *save* one subpacket, that is for a cycle that includes l vertices in $\hat{G}(\hat{V}, \hat{E})$ (that correspond to l clients), we transmit $l-1$ packets that satisfy all l clients. In total, αk subpackets will be saved,

---

however, it is possible use this algorithm to find optimal vertex-disjoint cycle packing by applying it to the vertex-split graph (as explained in Section B).

i.e., the total number of transmission is equal to $(n-\alpha)k$. This corresponds to saving α original packets. For each $v_i^j \in \hat{V}$ not included in any cycle in $\hat{C}$, transmit the corresponding subpacket p_i^j.

We proceed to establish the correctness of Algorithm *vSIC*.

Lemma 24 $n - OPT^f = OPT^f_{VCP}$.

Proof: First, we show that $n - OPT^f \leq OPT^f_{VCP}$. Consider an optimal vector-linear solution to Problem SIC. Let k be the number of subpackets in each packet with this solution. We note that the vector-linear solution with respect to the original packets is equivalent to the scalar-linear solution with respect to the subpackets. Then, by Lemma 21, it holds that $k(n - OPT^f)$ is less or equal to the maximum size of integer cycle packing in the fractional dependency graph described above. This, in turn implies that $k(n - OPT^f) \leq k(OPT^f_{VCP})$ or $n - OPT^f \leq OPT^f_{VCP}$.

We proceed to show that $n - OPT^f \geq OPT^f_{VCP}$. Consider an optimal fractional cycle packing $\psi(c)$ in the Dependency Graph (as defined in Definition 10, Section B). Let k be the minimum integer number for which it holds that $k\psi(c)$ is an integer for any $c \in C$. As we discussed above, this implies that there exists a integer cycle packing of size $k \cdot OPT^f_{VCP}$ in the fractional dependency graph. By Lemma 20 this implies that $k(n-OPT^f) \geq k \cdot OPT^f_{VCP}$ or $n-OPT^f \geq OPT^f_{VCP}$ and lemma follows. ■

Theorem 25 *Algorithm* vSIC *finds, in polynomial time, an optimal vector-linear solution to Problem SIC.*

Proof: By Lemma 24 it holds that $OPT^f = n - OPT^f_{VCP}$. Then, the theorem follows from the fact that the algorithm due to [40] finds an optimal solution to fractional cycle packing problem. ■

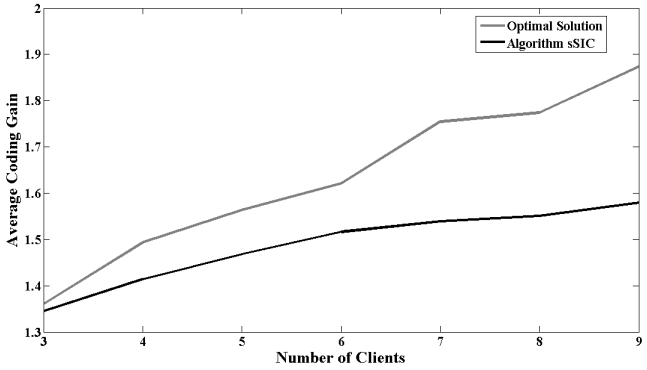

Fig. 40. Average *Coding Gain* versus number of clients for both the *Optimal solution* and the solution using algorithm *sSIC*

D. Performance Evaluation

In this section we evaluate the performance of the proposed scalar-linear solution for the *Sparse Index Coding* problem. More specifically, we evaluate the performance of the Algorithm *sSIC* presented in Figure 39, and compare it with the optimal linear solution to the *Index Coding* problem over $GF(2)$. Throughout this section the *Optimal solution* refers to the SAT-based time-efficient scalar-linear optimal solution over $GF(2)$ for the *Index Coding* problem as presented in Section IV.A.

The experimental setup is as follows. We consider n clients, where each client c_i requires packet p_i. The *has* set $H(c_i)$ for each client c_i is chosen randomly. More specifically, the cardinality ℓ_i of the *has* set $H(c_i)$ for each client c_i is selected from a uniform random distribution on integers $1, \ldots, n-1$. Then, we randomly choose ℓ_i packets for $H(c_i)$ out of n packets $\{p_1, \cdots, p_n\}$. Throughout this section each value in the simulation plots represents an average over 100.

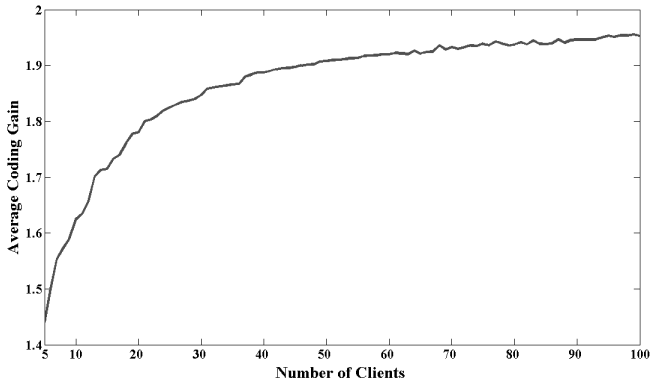

Fig. 41. Average *Coding Gain* versus number of clients for the solution using algorithm *sSIC*

Figure 40 shows the comparison of the algorithm *sSIC* and the optimal solution for the average coding gain versus the number of clients, where the *Coding Gain* is defined as the ratio between the minimum number of transmissions needed to satisfy all clients without encoding to the minimum number of transmissions required when scalar-linear coding is used. For example, for the instance of the *IC* problem shown in Figure 22 coding gain is $\frac{5}{3}$. The results show that on average the ratio of the average coding gains given by the Optimal solution and the algorithm *sSIC* differ by a factor less than 1.5. Hence, the presented solution is expected to perform well in practice as well. Table IV shows the comparison of the running time for the algorithm *sSIC* and the optimal solution. The comparison was performed over a Pentium 4 machine with 2.8 GHz processor. The results show that computing the optimal solution takes considerably more time than the proposed algorithm. Note that while finding the Optimal solution requires significant running time even for nine clients, the solution using the algorithm *sSIC* can be efficiently computed for hundred clients. Figure 41 shows the plot of the average coding gain versus number of clients for the solution

Table IV. Average CPU Time (in Seconds) Required by the Optimal Solution, and the Algorithm *sSIC*.

No. of Clients	**Optimal Solution**	**Algorithm *sSIC***
3	0.91	0.0026
4	0.7341	0.0034
5	0.9391	0.0042
6	0.9622	0.0049
7	3.579	0.0061
8	11.93	0.0071
9	82.97	0.0078

computed using the algorithm *sSIC*. The plot shows that for a large number of clients the proposed solution on average saves 49% of the transmissions compared to the solution without encoding.

E. Conclusion

In this chapter, we consider the *Sparse Index Coding* (SIC) problem. In this problem, each transmitted packet is a linear combination of at most two packets over a small field ($GF(2)$). This problem is important in practical settings due to low complexity of encoding and decoding.

We present both *scalar* and *vector* linear solutions for Problem SIC with provable performance guarantees. In particular, our algorithm yields a scalar-linear solution which has at most $2-\frac{1}{\sqrt{n}}$ more transmissions than the optimal. For the vector-linear case, we present an algorithm that yields an optimal solution. In addition, we show that finding an optimal solution for the scalar-linear case is an NP-complete problem.

We also present an extensive simulation study that demonstrate the advantages of the proposed solution in practical settings.

CHAPTER VII

DISTRIBUTED DATA RETRIEVAL*

In this chapter we consider the problem of accessing large data files stored at multiple locations across a content distribution, peer-to-peer, or massive storage network. We assume that the data is stored in either original form, or encoded form at multiple network locations. Clients access the data through simultaneous downloads from several servers across the network.

The central problem in this context is to find a set of disjoint paths of minimum total cost that connect the client with a set of servers such that the data stored at the servers is sufficient to decode the required file. We refer to this problem as the *Distributed Data Retrieval* (DDR) problem. We present an efficient polynomial-time solution for this problem that leverages the matroid intersection algorithm. Our experimental study shows the advantage of our solution over alternative approaches.

A. Introduction

In many practical settings, clients need to access large data files stored at multiple locations across the network. For example, in content distribution networks the data is stored across multiple geographical locations to enable efficient access by multiple clients. Similarly, in peer-to-peer networks clients retrieve popular files such as movies from their peers. In mass storage systems, the data is distributed throughout the network to increase the reliability and resilience to failures. When a client needs to obtain a copy of a large data object, it initiates simultaneous downloads from multiple

*Parts of this chapter are reprinted with permission from "An Optimal Solution to the Distributed Data Retrieval Problem" by M. A. R. Chaudhry, Z. Asad, and A. Sprintson in the proceedings of the 2010 IEEE Global Communications Conference (GLOBECOM), Miami, U.S.A., 2010, pages 1-6.

servers.

In this chapter, we consider the problem of accessing large data objects (such as multimedia files, datasets, etc.) stored at multiple network locations. We assume that each data object is divided into a number of fixed-size blocks, which are stored at servers across the network.

There are three major approaches for distributing the data across the servers. The first approach uses data replication or *mirroring*. With this approach, several copies of each block are stored on different servers across the network. A client needs to identify a subset of the nearby servers that collectively store all the required blocks and obtain one copy of each block through simultaneous downloads.

The second approach uses erasure correcting codes to generate parity check blocks. With this approach, k original blocks are encoded into n blocks using a *Maximum Distance Separable (MDS)* code, such that any k out of n blocks are sufficient for decoding the content of the file. A client needs to locate k nearby servers, and initiate simultaneous downloads to obtain k different coded blocks. The blocks are then decoded to obtain the content of the required file.

The third approach is to use a *general linear coding* scheme, which is not necessarily an MDS coding scheme. Such schemes are used, for example, in distributed storage systems [2]. In such schemes, a client needs to identify a subset of servers that collectively store enough data to be able to obtain the content of the original file. More specifically, suppose that the original file is divided into k blocks and that each block stored at a server is a linear combination of k original blocks. Thus, a client needs to simultaneously download data from k servers that store k linearly independent combinations of the original blocks. The contents of the original file can then be decoded by performing linear operations on the obtained data. Note that the general linear coding scheme includes the first two approaches as special cases.

In this chapter, we focus on the general linear coding setting and consider the problem of minimizing the total cost of downloading the contents of a file from multiple servers. We assume that each link in the network is associated with certain cost and has capacity constraints. Our goal is to find a set of k paths of minimum total cost that connect a subset of data servers with the client. The k paths should satisfy the following constraints:

1. Each path connects a data server and the client;
2. Each path is used for downloading a single data block;
3. The k downloaded data blocks are linearly independent;
4. The number of paths that share a single link cannot exceed the capacity of that link.

Note that the problem includes choosing a subset of data servers and the corresponding paths to the client through which the data will be downloaded. We refer to this problem as the *Distributed Data Retrieval* (DDR) problem.

Figure 42 demonstrates three approaches for storing three original blocks, a, b, and c across four servers and the corresponding instances of Problem DDR. Figure 42(a) shows a replication based approach. With this approach a client needs to find three disjoint paths, one originating at a server that stores block a, second at a server that stores block b, and third at a server that stores block c. Figure 42(b) demonstrates the approach where the data is stored using an *MDS* code (here, all operations are performed over $GF(2)$). With this approach, the client needs to find three disjoint paths of the minimum total cost to any three distinct servers. The general coding approach is depicted in Figure 42(c). In this scheme, the paths must originate

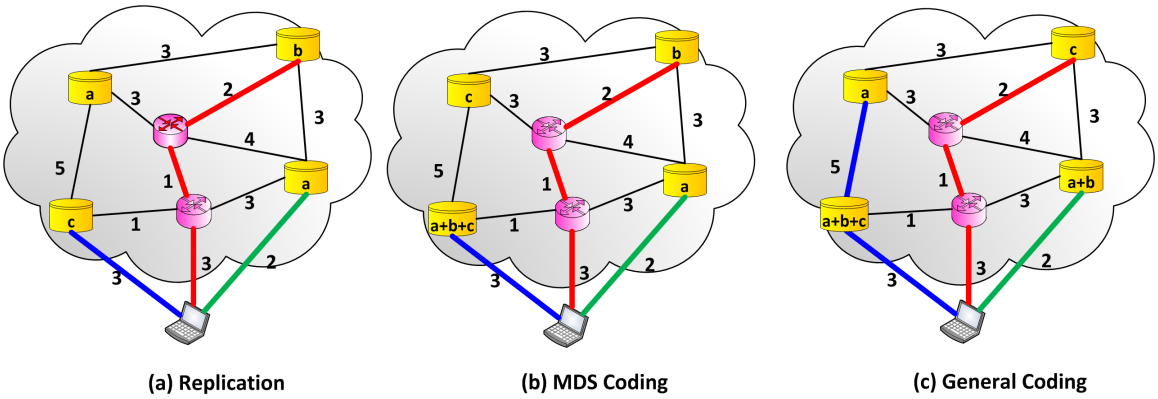

Fig. 42. Storage schemes considered in this chapter. (a) Replication-based approach. (b) An approach based on MDS codes. (c) An approach based on general linear codes. For each scenario, the optimal set of paths to retrieve packets a, b, and c is shown by thick lines.

at servers that store linearly independent combinations. In all figures, disjoint paths of minimum total cost are shown by thick lines.

Related Work. In [44] Suurballe and Tarjan presented a polynomial time algorithm for finding a set of edge disjoint paths of minimum total cost. The algorithm due to [44] can be used for solving Problem DDR in special cases. For example, if the data is encoded using an MDS code and each link has a unit capacity, the problem reduces to finding a minimum cost set of edge disjoint paths between any subset of storage nodes and the destination node (the size of the subset is equal to the number of blocks in the file). This can be accomplished by adding an auxiliary node s, connecting it to each storage node by an edge, and finding a set of disjoint paths between s and the destination node. Similarly, the algorithm due to [44] can be employed for solving Problem DDR when the mirroring or data replication approach is used. However, the algorithm [44] cannot be applied directly for the general case of Problem DDR.

Network coding solutions for the data retrieval problem were investigated by Dimakis et. al. [2]. The goal of this work is to minimize the total amount of data that need to be transferred to repair a failed storage server. References [45, 46] focused on network coding based content distribution, and considered the problem of minimizing the joint cost of transmission and storage for uncoded an coded cases. However, these works do not address the issue of selecting paths to transfer the data, which is the focus of this work.

Contributions. We introduce Problem DDR and present algorithms for its solution. First, in Section VII.D we consider a special case in which the network has a specific three-tier structure. Then, in Section VII.E we present a polynomial time algorithm for the general case. We also perform an experimental study that shows the advantage of our algorithms over alternative solutions.

B. Model

The communication network $\mathbb{N}$ is modeled by a directed graph $G(V, E)$ with the node set V and the edge set E. The network has n source (server) nodes $S = \{s_1, \ldots, s_n\}$ and a terminal node t. Without loss of generality, we assume that the sources nodes in S do not have incoming edges, while the terminal node t does not have outgoing edges.

We assume that terminal t needs to download a large file. The file is partitioned into h blocks $B = \{b_1, b_2, \ldots, b_h\}$, each block is an element of finite field $\mathbb{F}_q = GF(q)$. Each server node $s_i \in S$ stores a single linear combination x_i of blocks in B, i.e.,

$$x_i = \sum_{b_j \in B} \alpha_{ij} b_j,$$

where $\{\alpha_{ij}\}$ are elements of $GF(q)$. We also assume that the capacity of each edge

is one unit, i.e., it can transmit a single block per time unit. Note that this does not result in a loss of generality since a node of higher capacity can be represented by multiple nodes and an edge of higher capacity can be represented by multiple parallel edges.

We say that an edge $e(v, u)$ is *incident* to nodes v and u, and nodes v and u are incident to e. Each edge $e \in E$ is associated with a cost $c(e)$ which captures the cost of using this edge for transmitting a single block. The total cost of a path P in $G(V, E)$ is defined as the sum of the costs of its edges:

$$C(P) = \sum_{e \in P} c(e).$$

We consider the *Distributed Data Retrieval* (DDR) problem, defined as follows.

Problem DDR (Distributed Data Retrieval) *Find h edge-disjoint paths $\hat{P}_1, \hat{P}_2 \ldots, \hat{P}_h$ that connect a set of servers $\{s_{i_1}, s_{i_2}, \ldots, s_{i_h}\} \subseteq S$ with the terminal node t that satisfy the following conditions:*

1. *All the blocks in the set $\{x_{i_1}, x_{i_2}, \ldots, x_{i_h}\}$, stored at servers $s_{i_1}, s_{i_2}, \ldots, s_{i_h}$, are linearly independent;*
2. *The total cost of paths $\hat{P}_1, \hat{P}_2 \ldots, \hat{P}_h$ is less than or equal to any other set of paths that satisfy Condition (1).*

C. Preliminaries

Definition 26 *A* matroid $\mathcal{M}(X, \mathcal{I})$ *is an ordered pair formed by a* ground set X *and a collection $\mathcal{I}$ of subsets of X, that satisfy the following three conditions:*

1. $\emptyset \in \mathcal{I}$;
2. *If $Y \in \mathcal{I}$ and $Y' \subseteq Y$, then $Y' \in \mathcal{I}$;*

3. *If* $Y_1 \in \mathcal{I}$, $Y_2 \in \mathcal{I}$, $|Y_1| > |Y_2|$, *then there exists* $x \in Y_1 \backslash Y_2$ *such that* $Y_2 \cup \{x\} \in \mathcal{I}$.

Each $Y \in \mathcal{I}$ is referred to as an *independent set*. A maximal independent subset of X (with respect to inclusion) is referred to as a *base*. All bases of $\mathcal{M}$ have the same cardinality, referred to as the *rank* of $\mathcal{M}$.

Elements of X can be associated with a weight function $w : X \to \mathbb{Q}$. The weight of a subset Y of X is defined as the sum of weights of its elements:

$$w(Y) = \sum_{x \in Y} w(x).$$

One of the basic problems in matroid theory is to find the minimum-weight common base of two matroids $\mathcal{M}_1(X, \mathcal{I}_1)$ and $\mathcal{M}_2(X, \mathcal{I}_2)$, i.e., the subset Y of X of minimum weight such that Y is a base of both $\mathcal{M}_1$ and $\mathcal{M}_2$. The problem can be solved efficiently, in polynomial time. For a detailed description of matroid intersection algorithms see e.g., [47–49].

In our algorithms, we use concept of *integral network flows*.

Definition 1 (Integral Flow) *A integral* $(\hat{S}, t)$-flow f *is a binary function* $f : E \to \{0, 1\}$ *that satisfies the following two properties:*

1. *For all* $e(u, v) \in E$, *it holds that* $f_e \in \{0, 1\}$;
2. *For all* $v \in V \setminus \{S \cup \{t\}\}$, *it holds that*

$$\sum_{(u,v) \in E} f_{(u,v)} = \sum_{(v,u) \in E} f_{(v,u)}.$$

The *value* of a flow f is defined as follows:

$$|f| = \sum_{(v,t) \in E} f_{(v,t)} \tag{7.1}$$

An $(\hat{S}, t)$-flow is referred to as a *maximum flow* if it has the maximum value among all feasible $(\hat{S}, t)$-flows. An $(\hat{S}, t)$-flow of value $|f|$ can be decomposed into $|f|$ disjoint paths that connect nodes in $\hat{S}$ with t [50].

D. Algorithm for Three-tier Networks

In this section, we discuss a special case of Problem DDR in which the communication network $G(V, E)$ has a three-tier structure. This special case demonstrates the applications of the matroid intersection algorithm. We will generalize this approach in Section VII.E for general network topologies.

More specifically, the first tier consists of the set of source nodes $S = \{s_1, \ldots, s_n\}$, the second tier consists of set of intermediate nodes $V \setminus \{S \cup \{t\}\}$, and the third tier consists of the terminal t. Each edge in E either connects a tier 1 node and a tier 2 node, or a tier 2 node and the terminal t. An example of a three-tier network is shown in Figure 43.

We enumerate all paths $\{P_1, P_2, \ldots, P_l\}$ that connect sources $\{s_1, \ldots, s_n\}$ to the terminal t. For each path P_j we denote by $x(P_j)$ the block stored at the source node of P_j.

We define two matroids, $\mathcal{M}_1(\mathcal{P}, \mathcal{I}_1)$ and $\mathcal{M}_2(\mathcal{P}, \mathcal{I}_2)$ as follows:

- $\mathcal{P} = \{P_1, \ldots, P_l\}$ is the ground set of two matroids;
- $\mathcal{I}_1 \subseteq 2^{\mathcal{P}}$ is the collection of subsets of $\mathcal{P}$ which carry linearly independent blocks over $GF(q)$;
- $\mathcal{I}_2 \subseteq 2^{\mathcal{P}}$ is the collection of subsets of $\mathcal{P}$ such that each subset contains edge-disjoint paths.

Note that the above mentioned sets capture the basic constraints of Problem DDR, i.e., the constraint of selecting sources that host h linearly independent blocks is captured by $\mathcal{I}_1$, and the constraint on finding h edge-disjoint paths is captured by $\mathcal{I}_2$. Then, the set of h edge-disjoint paths, with least total cost that belongs to both $\mathcal{I}_1$ and $\mathcal{I}_2$, can be found using the minimum-weight common base algorithm [47,48]. The algorithm starts with a set $\bar{P} := \emptyset$ and iteratively augments it, such that at any time it holds that $\bar{P} = \mathcal{I}_1 \cap \mathcal{I}_2$. For the sake of completeness we describe the weighted matroid intersection algorithm, as presented in [48] below.

1. $\bar{P} := \emptyset$

2. Define a directed graph H with the node set $\mathcal{P}$, and the edge set as following. For any $P_i \in \bar{P}$ and $P_j \in \mathcal{P} \setminus \bar{P}$ add edges as follows:

   - If $(\bar{P} \setminus \{P_i\}) \cup \{P_j\} \in \mathcal{I}_1$ add an edge (P_i, P_j);
   - If $(\bar{P} \setminus \{P_i\}) \cup \{P_j\} \in \mathcal{I}_2$ add an edge (P_j, P_i).

3. Define sets:

$$\mathcal{P}_1 = \{P_i \in \mathcal{P} \setminus \bar{P} \mid \bar{P} \cup \{P_i\} \in \mathcal{I}_1\}$$

$$\mathcal{P}_2 = \{P_i \in \mathcal{P} \setminus \bar{P} \mid \bar{P} \cup \{P_i\} \in \mathcal{I}_2\}$$

4. For any node $P_i \in \mathcal{P}$ define its cost $l(P_i)$ by:

$$l(P_i) = -c(P_i) \text{ if } P_i \in \bar{P}$$

$$l(P_i) = c(P_i) \text{ if } P_i \notin \bar{P}$$

The cost of a path m in H, denoted by $c(m)$, is equal to the sum of the costs of the nodes traversed by m.

5. We consider two cases:

   Case 1: There exists a directed path m in H from a node in $\mathcal{P}_1$ to a node in $\mathcal{P}_2$

   - Choose the path m so that $c(m)$ is minimal and it has a minimum number of edges among all minimum cost paths from a node in $\mathcal{P}_1$ to a node in $\mathcal{P}_2$
   - Let the path m traverse the nodes $y_0, z_1, y_1, \ldots, z_g, y_g$ of H, in this order.

   $$\bar{P} := (\bar{P} \setminus \{z_1, \ldots, z_g\}) \cup \{y_0, \ldots, y_g\}$$

   - Go to Step 2

   Case 2: There is no directed path in the graph H from a node in $\mathcal{P}_1$ to a node in $\mathcal{P}_2$. Then, return $\bar{P}$.

Note that the set $\bar{P}$ returned by the algorithm is a maximum-cardinality common independent set.

Figure 43 demonstrates the execution of the algorithm on a three-tier instance of Problem DDR.

Proof of Correctness

First we show that $\mathcal{M}_1(\mathcal{P}, \mathcal{I}_1)$ and $\mathcal{M}_2(\mathcal{P}, \mathcal{I}_2)$ are valid matroids over ground set $\mathcal{P}$. Then, the correctness of our algorithm follows from that of the matroid intersection algorithm (see e.g., [48]).

First, we note that $\mathcal{M}_1$ is a *vector matroid* defined on the ground set $\mathcal{P}$ [51]. Next, we show that $\mathcal{M}_2$ is a matroid. We prove it by showing that it satisfies all

the properties of a matroid, as specified in Definition 26. The first condition follows from the fact that $\emptyset \in \mathcal{I}_2$. A subset of a set of disjoint paths also contains disjoint paths, which implies the second condition. To show the third condition, consider two sets, $Y_1 \subseteq \mathcal{P}$ and $Y_2 \subseteq \mathcal{P}$, such that $|Y_1| \geq |Y_2|$. Note that the paths in Y_2 use $|Y_2|$ intermediate nodes, while paths in Y_1 use more than $|Y_2|$ intermediate nodes. Thus, there exists at least one path P' in Y_1 that uses a different intermediate node than the paths in Y_2. This path does not share edges with any path in Y_2. Thus, it holds that $Y_2 \cup \{P'\}$ is a set of edge-disjoint paths in $\mathcal{I}_2$

E. Algorithm for General Networks

In this section, we describe our algorithm for Problem DDR in general networks. Our algorithm includes the following steps:

1. Construct an auxiliary bipartite graph $H(\hat{V}_1, \hat{V}_2, \hat{E})$ such that a flow of value h between nodes in S and terminal t corresponds to a maximum matching in H.

2. Construct two matroids, $\mathcal{M}_1(\hat{E}, \mathcal{I}_1)$ and $\mathcal{M}_2(\hat{E}, \mathcal{I}_2)$. The matroids capture the matching constraints as well as the linear independence constraints imposed on the source nodes.

3. Find the minimum-weight common base $\hat{E}'$ of the matroids $\mathcal{M}_1(\hat{E}, \mathcal{I}_1)$ and $\mathcal{M}_2(\hat{E}, \mathcal{I}_2)$. The set $\hat{E}' \subseteq \hat{E}$ is a maximum matching in $H(\hat{V}_1, \hat{V}_2, \hat{E})$.

4. Find a set of h disjoint paths $\{\hat{P}_1, \hat{P}_2 \ldots, \hat{P}_h\}$ in $G(V, E)$ that corresponds to matching $\hat{E}'$ in $H(\hat{V}_1, \hat{V}_2, \hat{E})$. Paths $\{\hat{P}_1, \hat{P}_2 \ldots, \hat{P}_h\}$ connect h sources $\{s_{i_1}, s_{i_2}, \ldots, s_{i_h}\}$ in S to the destination node t.

1. Constructing the Bipartite Graph $H(\hat{V}_1, \hat{V}_2, \hat{E})$

We use a reduction from flow network described by graph $G(V, E)$ to an instance $H(\hat{V}_1, \hat{V}_2, \hat{E})$ of the bipartite matching problem [20, 52].

Given a graph $G(V, E)$, source nodes $S = \{s_1, \ldots, s_n\}$ and the destination node t we construct the auxiliary graph $H(\hat{V}_1, \hat{V}_2, \hat{E})$ as follows. First, for each edge $e \in E$, we add node $\hat{v}_e^1$ to $\hat{V}_1$ and a node $\hat{v}_e^2$ to $\hat{V}_2$. Next, for each source $s_i \in S$ we add a corresponding node $\hat{s}_i$ to $\hat{V}_2$. Next, we add h destination nodes $\hat{t}_1, \ldots, \hat{t}_h$ to $\hat{V}_1$. Thus,

$$\hat{V}_1 = \{\hat{v}_e^1 \mid e \in E\} \cup \{\hat{t}_1, \ldots, \hat{t}_h\}$$

and

$$\hat{V}_2 = \{\hat{v}_e^1 \mid e \in E\} \cup \{\hat{s}_1, \ldots, \hat{s}_n\}.$$

Next, we construct the edge set $\hat{E}$ of H as follows:

1. For each edge $e \in E$, we add an edge $(\hat{v}_e^1, \hat{v}_e^2)$ of zero cost.

2. For each node $v \in V$, do:

   - For each pair of edges e' and e'' such that e' is an incoming edge of v and e'' is an outgoing edge of v we add an edge $(\{\hat{v}_{e'}^2, \hat{v}_{e''}^1)$ of cost $(c(e') + c(e''))/2$.

3. For each outgoing edge e' of a source node $s_i \in S$ add an edge $(\hat{s}_i, \hat{v}_{e'}^1)$ of cost $c(e')/2$.

4. For each incoming edge e' of t add an edge $(\hat{s}_i, \hat{v}_{e'}^1)$ of cost $c(e')/2$.

Figure 44(a) demonstrates the construction of graph $H(\hat{V}_1, \hat{V}_2, \hat{E})$.

Karp et. al. [52] showed that a maximum matching in $H(\hat{V}, \hat{E})$ yields a maximum flow in $G(V, E)$ according to the following rule: edge e carries a flow of value one if and only if one of the the following conditions hold:

- Node $\hat{v}_e^1$ is matched with some node $\hat{v}_{e'}^2$ such that e' is an incoming edge of the head node of e;
- Node $\hat{v}_e^2$ is matched with some node $\hat{v}_{e''}^1$, such that e is an incoming edge of the head node of e''.

Lemma 27 *A flow of size h between a subset of nodes in S and a destination node t corresponds to a maximum matching in $H(\hat{V}, \hat{E})$ of the same cost. Furthermore, a maximum matching in $H(\hat{V}, \hat{E})$ yields a corresponding flow in $G(V, E)$ of value h between a subset of S and t of the same cost.*

Proof: Follows from the construction of graph $H(\hat{V}, \hat{E})$. ■

2. Matroid Definition

We proceed to define two matroids, $\mathcal{M}_1(\hat{E}, \mathcal{I}_1)$ and $\mathcal{M}_2(\hat{E}, \mathcal{I}_2)$. Both matroids are defined over the ground set of edges in $\hat{E}$. We define $\mathcal{I}_1$ be a collection of subsets $I \subseteq \hat{E}$ such that for each node $\hat{v} \in \hat{V}_1$ at most one edge incident to $\hat{v}$ belong to I.

Next, we define $\mathcal{I}_2$ be a collection of subsets $I \subseteq \hat{E}$ that satisfy the following constraints:

1. For each node $\hat{v} \in \hat{V}_2$ at most one edge incident to $\hat{v}$ belongs to I.
2. Let $S' = \{\hat{s}_{i_1}, \ldots, \hat{s}_{i_l}\}$ be a subset of $\{\hat{s}_1, \ldots, \hat{s}_n\}$ such that each node $\hat{s}_i \in S'$ has an edge in I incident to it. Then, the set of linear combinations $\{x_{i_1}, \ldots, x_{i_l}\}$ stored at $S' = \{\hat{s}_{i_1}, \ldots, \hat{s}_{i_l}\}$ is of rank l.

Lemma 28 *Matroids $\mathcal{M}_1(\hat{E}, \mathcal{I}_1)$ and $\mathcal{M}_2(\hat{E}, \mathcal{I}_2)$ are valid matroids of rank $|E| + h$.*

Proof: It is easy to verify that set $\mathcal{I}_1$ satisfies the three conditions of Definition 26. The rank of $\mathcal{M}_1$ is equal to the cardinality of $\hat{V}_1$, i.e., $|E| + h$.

Similarly, it is easy to see that $\mathcal{I}_2$ satisfies that first two conditions of Definition 26. To show the third condition, we divide set $\hat{V}_2$ into two sets $\hat{V}_2^1 = \{\hat{v}_e^1 \mid e \in E\}$ and $\hat{V}_2^2 = \{\hat{s}_1, \ldots, \hat{s}_n\}$. Let Y_1 and Y_2 be two elements of $\mathcal{I}_2$ such that $|Y_1| > |Y_2|$. Then, at least one of the following two statements hold:

- The number of edges in Y_1 incident to nodes in $\hat{V}_2^1$ is strictly larger than the number of edges in Y_2 incident to nodes in $\hat{V}_2^1$. In this case, one of the edges e in Y_1 incident to a node in $\hat{V}_2^1$ can be added to Y_2, such that $Y_2 \cup \{e\} \in \mathcal{I}_2$.
- The number of edges in Y_1 incident to nodes in $\hat{V}_2^2$ is strictly larger than the number of edges in Y_2 incident to nodes in $\hat{V}_2^2$. Then, the set of linear combinations stored at nodes in $\hat{V}_2^2$ incident to Y_1 has a higher rank than the set of linear combinations stored at nodes in $\hat{V}_2^2$ incident to Y_2. Thus, in this case, one of the edges e in Y_1 incident to a node in $\hat{V}_2^2$ can be added to Y_2, such that $Y_2 \cup \{e\} \in \mathcal{I}_2$.

■

3. Finding Disjoint Paths

The next step is to find the minimum-weight common base $\hat{E}'$ of two matroids $\mathcal{M}_1(\hat{E}, \mathcal{I}_1)$ and $\mathcal{M}_2(\hat{E}, \mathcal{I}_2)$. This can be done efficiently, in polynomial time, using a standard matroid intersection algorithm (see e.g., [48, 53]). Note that $\hat{E}' \subseteq \hat{E}$ is a matching of $H(\hat{V}, \hat{E})$. This is due to the matching constraints are imposed by matroids $\mathcal{M}_1$ and $\mathcal{M}_2$. Also, the size of $\hat{E}'$ is equal to the size of the set $\hat{V}_1$, hence $\hat{E}'$ is a maximum matching. Note also, that exactly h nodes in $\{\hat{s}_1, \ldots, \hat{s}_n\}$ have an edge in $\hat{E}'$ incident to them. We denote these nodes by $\{\hat{s}_{i_1}, \ldots, \hat{s}_{i_h}\}$.

The final step is to transform the maximum matching $\hat{E}'$ of $H(\hat{V}, \hat{E})$ into a set of h edge-disjoint paths $\{\hat{P}_1, \hat{P}_2 \ldots, \hat{P}_h\}$ that connect a subset of sources $\{s_{i_1}, s_{i_2}, \ldots, s_{i_h}\}$

in S to the destination node t by using the method described in Section VII.1.

We summarize our discussion by the following theorem.

Theorem 29 *The algorithm described above finds, in polynomial time, an optimal solution to Problem DDR.*

Proof: Follows from lemmas 27 and 28 and the correctness of the matroid intersection algorithm. ■

F. Numerical Results

In order to evaluate the performance of the proposed solution, we have used six practical ISP topologies of the backbone networks from the *Rocketfuel* project [22]. For the purpose of simulations each backbone ISP map is transformed into a graph where each backbone router is represented by a node, and a link between any pair of backbone routers is represented by an edge. The cost assigned to each edge is equal to the corresponding link weight inferred by *Rocketfuel*. The approximation of link weights is based on end-to-end measurements [54].

In each experiment we start by choosing a random set of n sources and a terminal node from an ISP topology. Then we create a file that contains r blocks. We assign a random linear combination of these blocks to each source. We then find a solution to Problem DDR by using two different techniques. First technique uses the algorithm presented in Section VII.E. The second technique relies on the following greedy solution.

1. Let S be the set of all subsets of r sources,
2. For each set $S_i \in S$ in an arbitrary order:
    - If rank of the blocks assigned to S_i is r then:

– Find r edge disjoint paths. If paths exist, return these paths, otherwise, go to Step 2.

We define the performance metric, referred to as *gain*, as the ratio of the cost of the solution obtained through the heuristic and the cost of the solution presented in Section VII.E. We performed 1000 random experiments on each of six ISP topologies. The results in Figure 45 show an average gain of about 1.6. Furthermore, Figure 46 shows the number of attempts that the greedy heuristic makes in order to get first feasible solution is on average about 3. Note that algorithm presented in Section VII.E always finds the least cost optimal solution in one attempt.

G. Conclusion

This chapter focus on the problem of distributed data retrieval where the data is distributed across different servers using a combination of replication and coding. The objective is to connect the terminal with the subset of sources hosting linearly independent packets using the least cost paths. We present a simple and intuitive algorithm to find an optimal solution for DDR based on matroid intersection algorithm that works for a subclass of networks. The algorithm requires an explicit knowledge of the paths from the sources to the terminal. In addition to this, we present an efficient polynomial time algorithm for the DDR problem that does not need an explicit knowledge of paths and works for general networks. Our experimental study show the advantage of the presented algorithm over greedily selecting data sources for data retrieval.

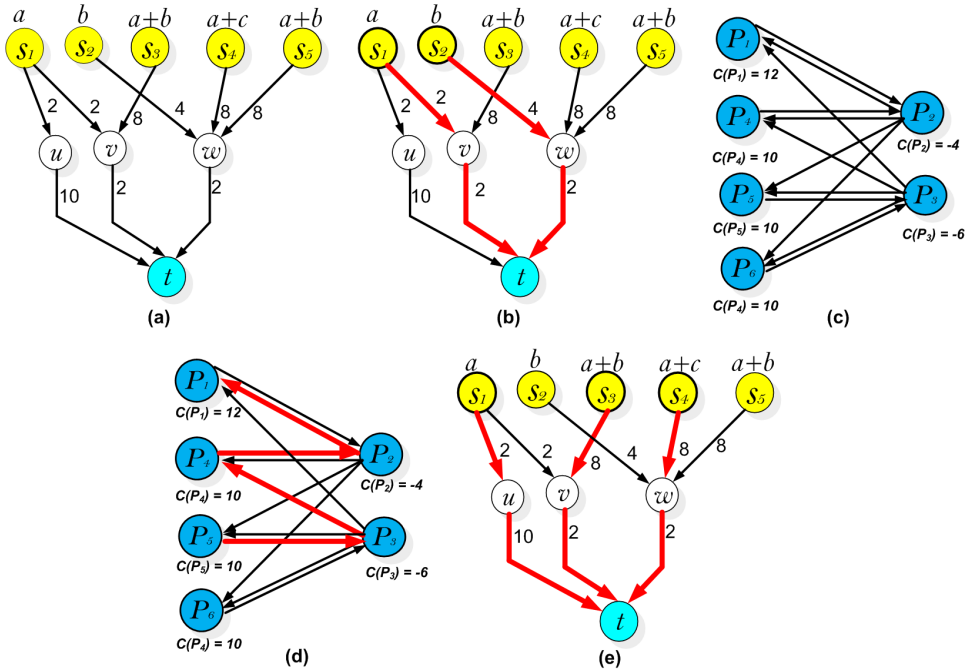

Fig. 43. Execution of the algorithm for three-tier networks (a) A three-tier instance of Problem DDR with five sources $s_1, \ldots, s_5$ hosting blocks x_1=a, x_2=b, x_3=a+b, x_4=a+c, x_5=a+b respectively. There are six paths from the sources to the terminal: P_1=$\{s_1, u, t\}$, P_2=$\{s_1, v, t\}$, P_3=$\{s_2, w, t\}$, P_4=$\{s_3, v, t\}$, P_5=$\{s_4, w, t\}$, P_6=$\{s_5, w, t\}$ with the corresponding costs as $c(P_1)$=12, $c(P_2)$=4, $c(P_3)$=6, $c(P_4)$=10, $c(P_5)$=10, $c(P_6)$=10, and the corresponding blocks $x(P_1)$=x_1, $x(P_2)$=x_1, $x(P_3)$=x_2, $x(P_4)$=x_3, $x(P_5)$=x_4, $x(P_6)$=x_5 respectively. (b) Path $\bar{P}$=$\{P_2,\ P_3\}$ selected in the second iteration (shown in bold). (c) The directed graph H constructed in the third iteration of the proposed algorithm. (d) The minimum cost path $m = \{P_5,\ P_3,\ P_4,\ P_2,\ P_1\}$ from $\mathcal{P}_1$=$\{P_5\}$ to $\mathcal{P}_2$=$\{P_1\}$ (shown in bold), $c(m) = 22$. (e) The optimal solution $\bar{P}$=$\{P_1, P_4, P_5\}$ (shown in bold).

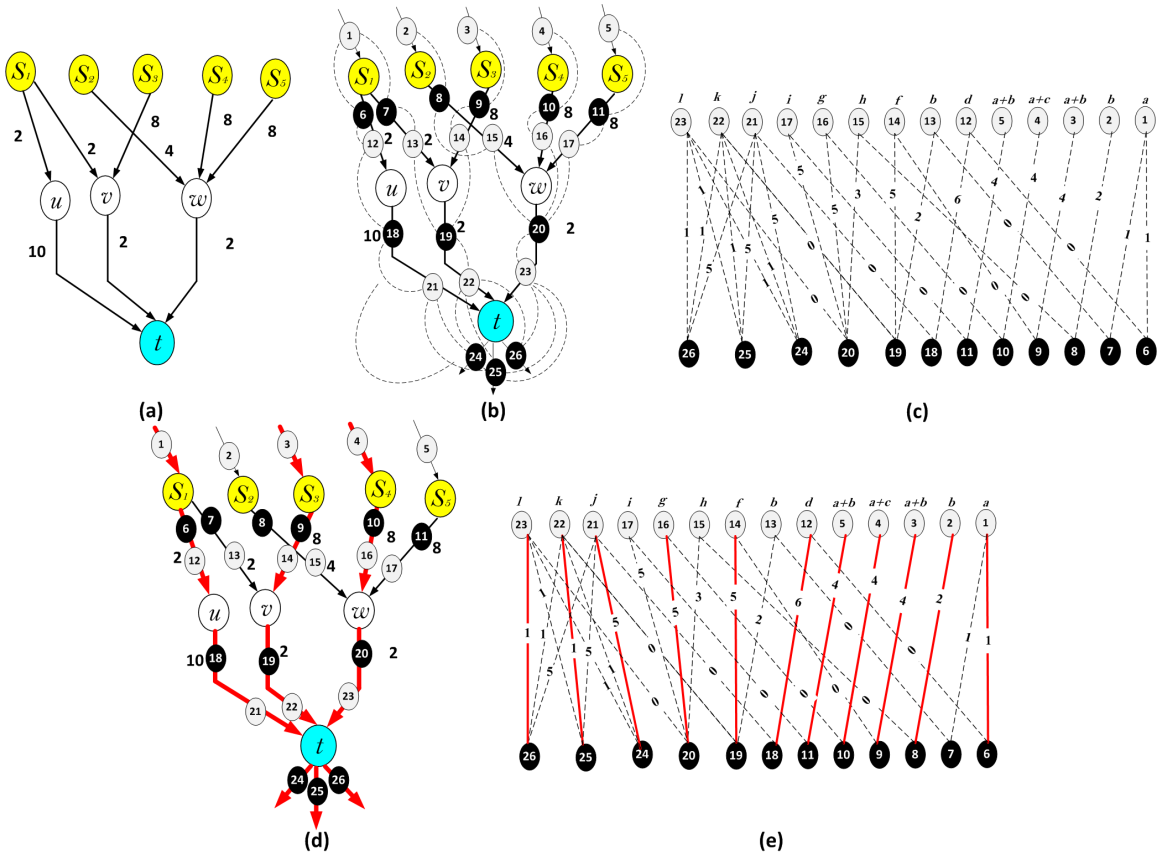

Fig. 44. Execution of the algorithm for general networks. (a) A general instance for Problem DDR. (b) Construction of an auxiliary graph $H(\hat{V}_1, \hat{V}_2, \hat{E})$ (nodes in $\hat{V}_1$ are black and nodes in $\hat{V}_2$ are shown in white). (c) Bi-partite graph $H(\hat{V}_1, \hat{V}_2, \hat{E})$. (d) Maximum matching in $H(\hat{V}_1, \hat{V}_2, \hat{E})$. (e) An optimal solution to Problem DDR.

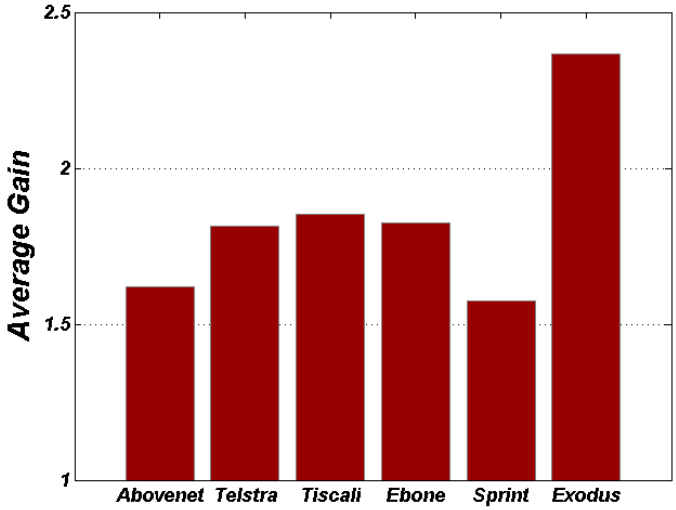

Fig. 45. Simulation results for *Gain* for six ISP backbone topologies given by Rocketfuel [22].

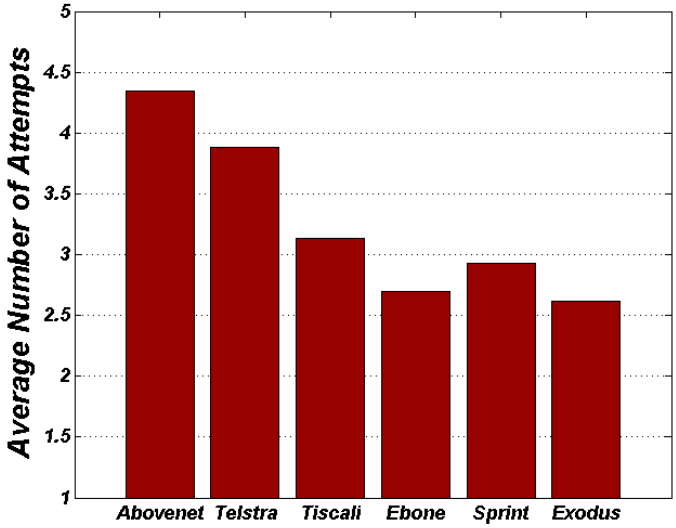

Fig. 46. Simulation results for number of iterations taken by heuristic to find a feasible solution for six ISP backbone topologies given by Rocketfuel [22].

CHAPTER VIII

EFFICIENT REROUTING ALGORITHMS FOR CONGESTION MITIGATION*

In the advent of smaller devices, a significant increase in the density of on-chip components has raised congestion and overflow as critical issues in VLSI physical design automation. In this chapter, we present novel techniques for reducing congestion and minimizing overflows. Our methods are based on ripping-up nets that go through the congested areas and replacing them with *congestion-aware* topologies.

Our contributions can be summarized as follows. First, we present several efficient algorithms for finding *congestion-aware* Steiner trees, i.e., trees that avoid congested areas of the chip. Next, we show that the novel technique of *network coding* can lead to further improvements in routability, reduction of congestion, and overflow avoidance. Finally, we present an algorithm for identifying efficient congestion-aware network coding topologies. We evaluate the performance of the proposed algorithms through extensive simulations.

A. Introduction

In almost any VLSI design flow, global routing is an essential stage that determines the signal interconnections. Therefore, the capability of the global router may significantly affect the design turn-around time. Moreover, the results of the global routing stage impact many circuit characteristics, such as power, timing, area, and signal integrity. Global routing poses major challenges in terms of the efficient computation of quality routes. In fact, most of the global routing problems, even special cases,

*Parts of this chapter are reprinted with permission from "Efficient Congestion Mitigation Using Congestion-Aware Steiner Trees and Network Coding Topologies" by M. A. R. Chaudhry, Z. Asad, A. Sprintson, and J. Hu VLSI Design, vol. 2011, Article ID 892310, 9 pages, 2011.

tend to be NP-complete [55], [56].

In the advent of smaller devices, a significant increase in the density of on-chip components results in a larger number of nets that need to be routed, which, together with more stringent routing constraints, results in increasing congestion and overflow. In this chapter, we propose novel techniques for congestion avoidance and overflow reduction. Our methods are designed for the rip-up-and-reroute phase of the global routing stage. At this stage, all the nets have already been routed using a standard pre-routing technique, however some of the nets need to be rerouted due to high congestion and overflow. Our methods are based on ripping-up nets that go through congested areas and replacing them with congestion-aware topologies. The proposed techniques facilitate even distribution of the routing load along the available routing areas. We propose efficient algorithms for finding congestion-aware Steiner trees that favor uncongested routes. In addition, we use the novel technique of *network coding* for further reduction of congestion and overflow avoidance.

1. Congestion-aware Steiner Trees

The major goal of congestion-aware Steiner tree routing is to find a tree that connects the required set of nodes (pins of a net) while avoiding congested areas with a minimum penalty in terms of the total wirelength. In addition, the running time of the routing algorithm should scale well with the growing number of nets. These requirements pose several challenges in terms of the algorithm design. The first challenge is to select a cost function that adequately captures the local congestion conditions at the edges of the routing graph. Next, the algorithm should find a minimum cost tree within acceptable running time. Since finding a Steiner tree is an NP-complete problem, the algorithm needs to use an approximation scheme or employ a heuristic approach. Finally, the proposed algorithm should ensure that the overall performance

of the rip-up-and-reroute phase is satisfactory in terms of congestion mitigation and overflow reduction. In this chapter we evaluate several cost functions which take into account various factors such as wire density, overflow, and congestion history. We propose several efficient algorithms for Steiner tree routing and compare their performance. Our algorithms are based on known approximations for the Steiner tree problem, heuristic methods, and artificial intelligence techniques.

2. Network Coding

The basic idea of the *network coding* technique [1] is to enable the intermediate nodes to generate new signals by combining the signals arriving over their incoming wires. This is in contrast to the standard approach, in which each node can only forward or duplicate the incoming signals.

For example, consider a routing instance depicted in Figure 47(a). In this example, we need to route two nets, one connecting source s_1 with terminals t_1, t_2, and t_3, and the other connecting source s_2 with the same set of terminals. The underlying routing graph is represented by a grid as shown in Figure 47(a). Suppose that due to congestion each edge of this graph has a residual capacity of one unit, i.e., each edge can accommodate only a single wire. It is easy to verify that using traditional Steiner tree routing only one net can be routed without an overflow. For example, Figure 47(b) shows a possible routing of a net that connects s_1 with terminals t_1, t_2, t_3. In contrast, Figure 47(c) shows that routing of both nets results in an overflow. In this example, two nets transmit different signals, a and b, over separate Steiner trees. Figure 47(d) shows that the network coding approach allows to route both nets without overflows. With this approach, the terminal t_1 creates a new signal, $a \oplus b$, which is delivered to terminals t_2 and t_3, while the signals a and b are delivered to terminals t_2, and t_3 directly. It is easy to verify that each terminal can decode the two original

symbols a and b.

The network coding technique offers two distinct advantages. First, it has a potential of solving difficult cases that cannot be solved using traditional routing. For example, for the routing instance shown in Figure 47 the traditional routing results in an overflow of value 1 whereas with the network coding technique there are no overflows. Second, the network coding technique can decrease the total wirelength. For example, in the routing instance shown in Figure 47 the total wirelength for the traditional routing solution is 8 whereas for the network coding solution the total wirelength is 7.

3. Previous Work

In the past decades, researchers have strived to improve the performance of global routing algorithms (see e.g., [57], [58], [59], and references therein). To handle the complexity of large scale global routing, multilevel routing techniques are proposed in [60] and [61]. Recently proposed BoxRouter [62,63] is based on progressive Integer Linear Programming (ILP) and rip-up-and-reroute techniques. A fast routing method is presented in [64]. Reference [65] proposes an approach based on the Lagrangian multiplier technique. An effective edge shifting technique is presented in [66]. Most of these previous works adopt the rip-up-and-reroute strategy. However, they usually reroute one path (i.e., a 2-pin connection) at a time. In contrast, our method reroutes entire multi-pin nets. We also propose to use network coding techniques to further reduce congestion and eliminate overflows.

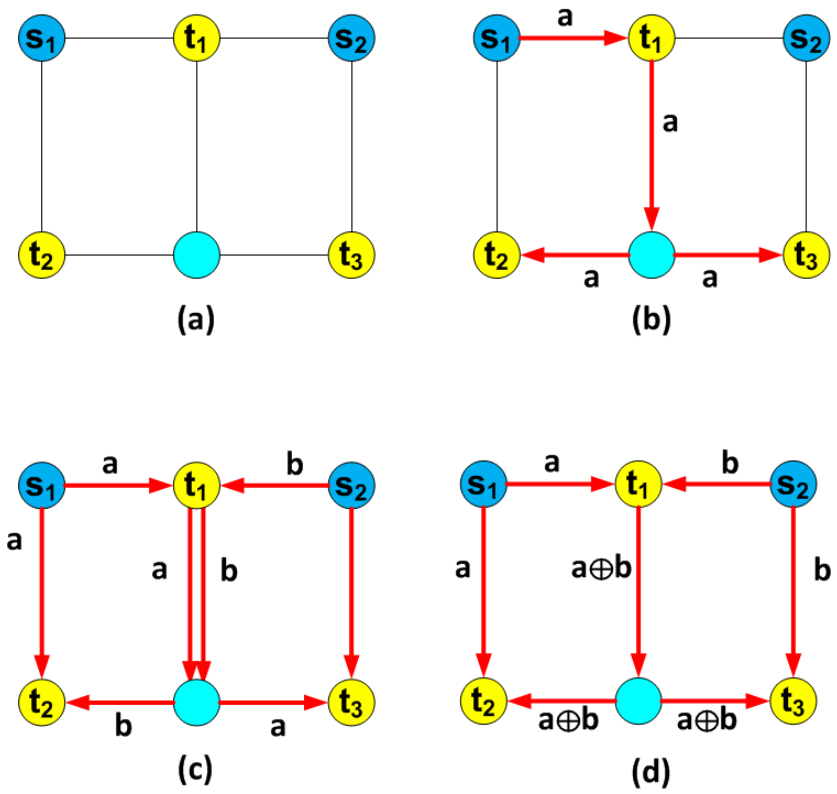

Fig. 47. (a) The underlying routing graph with two source nodes s_1, s_2, and three terminals t_1, t_2, t_3. (b) A rectilinear Steiner tree that connects source s_1 to all terminals. (c) Two rectilinear Steiner trees that connect sources s_1 and s_2 to all terminals. (d) A network coding solution.

4. Contribution

The chapter makes the following contributions. First, we propose several algorithms for finding efficient congestion-aware Steiner trees. Second, we show that the novel technique of network coding can lead to further improvements in routability, reduction of congestion, and overflow minimization. Finally, we provide an algorithm for identifying efficient congestion-aware network coding topologies. We evaluate the performance of the proposed algorithms through extensive simulations.

B. Model

In this chapter, we adopt the most commonly used global routing model. The routing topology is represented by a grid graph $G(V, E)$, where each node $v \in V$ corresponds to a global routing cell (GCell) [63, 65] and each routing edge $e \in E$ corresponds to a boundary between two adjacent GCells. A set $S = \{n_1, n_2, \ldots, n_{|S|}\}$ of nets are to be routed on this graph. Each net $n_i \in S$ connects a source node s_i with terminal nodes $T_i = \{t_1, t_2, \ldots, t_{|T_i|}\}$. If there is a wire connection between two adjacent GCells, the wire must cross their boundary and utilize the corresponding routing edge. Each routing edge $e \in E$ has a certain routing capacity $c(e)$ which determines the number of wires that can pass through this edge. We denote by $\eta(e)$ the number of wires that are currently using edge e.

1. Global Routing Metric

The goal of a global router is to minimize congestion. Some of the important metrics for a global router are defined as follows:

- **Overflow:** For each edge $e \in E$, the overflow $ov(e)$ of e is defined as

$$ov(e) = \begin{cases} \eta(e) - c(e) & \text{if } \eta(e) > c(e) \\ 0 & \text{otherwise.} \end{cases}$$

  The maximum overflow $ov_{\max}$ is defined as

$$ov_{\max} = \max_{e \in E} ov(e).$$

  Total overflow ov_{tot} is defined as

$$ov_{tot} = \sum_{e \in E} ov(e).$$

- **Wirelength:** Total wire-length $wlen$ is defined as

$$wlen = \sum_{e \in E} \eta(e).$$

- **Density:** The density $d(e)$ of edge $e \in E$ is defined as

$$d(e) = \frac{\eta(e)}{c(e)}.$$

2. Cost Functions

Our algorithms associate each edge $e \in E$ in the graph with a cost function $\rho(e)$ which captures its congestion and overflow. The cost of the tree is defined as the sum of the costs of all of its edges. Our goal is to identify trees that go through congested areas and replace them by Steiner trees or network coding topologies that go through areas with low congestion.

In this work we consider several cost functions, described below.

Polynomial Cost Function. We propose a cost assignment function where the cost of an overflowed edge is a polynomial function of the sum of its density and overflow. Formally, our proposed cost function is defined as follows:

$$\rho(e) = (d(e) + ov(e))^{\alpha}, \tag{8.1}$$

where α is a constant which determines the relative penalty for the congested edges.

Exponential Cost Function. We use the cost assignment function proposed by [65]. With this cost assignment, the cost of an edge is an exponential function of its density:

$$\rho(e) = \begin{cases} \mathrm{Exp}(\beta \cdot (\mathrm{d(e)} - 1)) & \text{if } d(e) > 1 \\ d(e) & \text{otherwise,} \end{cases} \tag{8.2}$$

where β is a constant which determines the penalty for overflowed edges.

History-based Cost Function. This cost function assigns cost to an edge based on its congestion history [63, 65]. Specifically, each edge is associated with a parameter h_e that specifies the number of times the edge has been overflowed during the previous iterations of the algorithm. That is, each time the edge with an overflow is used, the parameter h_e is incremented by one. Then, the modified cost $\rho'(e)$ of the edge is defined as follows:

$$\rho'(e) = 1 + h_e \cdot \rho(e). \tag{8.3}$$

Here, $\rho(e)$ is either the polynomial cost function (Equation (8.1)) or exponential cost function (Equation (8.2)). If the density of the edge is less than or equal to one, the parameter h_e is initially set to zero.

Since we focus on the rerouting phase, we assume that for each net $n_i \in S$ there exists a Steiner tree ϕ_i which connects all nodes in n_i. Given a set of trees $\{\phi \,|n_i \in S\}$ we can determine the values of $ov(e)$, $\eta(e)$ for each edge $e \in E$ and identify the set of *congested nets* $S' \subseteq S$. A net $n_i \in S'$ is referred to as *congested* if its Steiner tree ϕ_i has at least one edge with overflow.

We propose a two-phase solution for rerouting congested nets using congestion-aware topologies. In the first phase we iteratively rip-up each net $n_i \in S'$ and reroute it using a congestion-aware Steiner tree with the goal of minimizing the maximum overflow $ov_{\max}$ and the total overflow ov_{tot}. In second phase we deal with the nets that remain congested at the end of the first phase and rip-up-and-reroute pairs of congested nets using congestion-aware network coding topologies to further reduce congestion and minimize the number of overflows. Note that the nets considered in phase two correspond to the difficult cases where congestion avoidance was not possible even after several attempts of ripping-up and rerouting *individual* nets. Therefore in the second phase we consider the *pairs* of congested nets for further improvement.

Example given in Figure 47 shows the advantage of using network coding topologies for routing pairs of nets over using standard routing techniques that handle each net separately.

C. Congestion-Aware Steiner Trees

In this section we present several techniques for finding congestion-aware Steiner trees. Our goal is to find Steiner trees that minimize congestion with a minimum penalty in terms of the overall wirelength. We would like to achieve better tradeoffs between congestion mitigation and total wirelength. These tradeoffs are useful for practical applications, as in some cases congestion mitigation is preferable to wirelength reduction, whereas in other cases the wirelength reduction is of higher priority.

1. Previous Work on Steiner Tree Routing

The Steiner tree problem is a well studied NP-complete problem [18]. There is a wealth of heuristic and approximate solutions that have been developed for this problem. The best known approximation algorithm has an approximation ratio of 1.55 (i.e., the cost of the tree returned by the algorithm is less than 1.55 times the optimum) [67]. The best known approximations require significant computation time so we focus on computationally feasible and easy to implement approximation and heuristic solution for constructing Steiner trees.

2. Algorithms for Finding Congestion-Aware Trees

As mentioned above, our goal is to rip-up and reroute nets that use congested edges of $G(V, E)$. For each net $n_i \in S$ which has been ripped up, we need to find an

alternative Steiner tree that uses uncongested routes. In this section we describe five algorithms for finding congestion-aware Steiner trees. The first three algorithms use combinatorial techniques (see e.g., [68], [55], [69]) while the last two are based on the intelligent search techniques [70]. The performance of the algorithms is evaluated in Section E.

a. Algorithm $stTree1$

This algorithm approximates a minimum cost Steiner tree by using a *shortest path tree*. A shortest path tree is a union of the shortest paths between source s_i and a set of terminals T_i. A shortest path tree can be identified by a single invocation of Dijkstra's algorithm. However, the cost of the tree may be significantly higher than the optimum.

b. Algorithm $stTree2$

This algorithm constructs the tree in an iterative manner. We iteratively process the terminals in T_i in the increasing order of their distance form s_i. More specifically, we first find a shortest path P_1 between source s_i and terminal t_1. Then, we assign a zero cost to all edges that belong to P_1 and find a shortest path P_2 between s and t_2 with respect to modified costs. The idea behind this algorithm is to encourage sharing of the edges between different paths. That is, if an edge e belongs to P_1, it can be used in P_2 with no additional cost. In general, when finding a shortest path to terminal t, all edges that belong to paths of previously processed terminals are assigned a zero cost. This algorithm requires $|T|-1$ iterations of Dijkstra's algorithm, but it typically returns a lower cost tree than Algorithm $stTree1$.

c. Algorithm $stTree3$

This is a standard approximation algorithm with the approximation ratio of 2 (i.e., the cost of the tree returned by the algorithm is at most two times higher than the optimal cost). Specifically, with this algorithm we find a shortest path between each pair of nodes in the set $\bar{T} = \{T \cup \{s_i\}\}$. Then, we construct a complete graph G' such that each node in G' corresponds to a node in $\bar{T}$. The weight of an edge $e \in G'$ is equal to the minimum length of the path between two corresponding nodes in $\bar{T}$. The algorithm then finds a minimum spanning tree ϕ in G'. Next, each edge in ϕ is substituted by the corresponding shortest path in G which results (after removing redundant edges) in Steiner tree in G that connects source s_i with terminals in T_i.

d. Algorithm $stTree4$

Algorithm $stTree4$ is an intelligent search based algorithm. Our approach is inspired by Algorithm A^*. Algorithm A^* is a shortest path algorithm that uses a heuristic function $\lambda(v)$ to determine the order of visiting nodes of the graph, in order to improve its running time. Specifically, for each node v we define $\lambda(v)$ to be the maximum distance between node v and a terminal $t \in T_i$ which has not yet been visited. The distance between v and $t \in T_i$ is defined as the minimum number of hops that separate v and t in G. The Algorithm $stTree4$ follows the same steps as Algorithm $stTree1$ but it uses Algorithm A^* with heuristic function $\lambda(v)$ to find shortest paths.

e. Algorithm $stTree5$

Algorithm $stTree5$ is also based on Algorithm A^*. It follows the same steps as Algorithm $stTree2$, but it uses Algorithm A^* with the same heuristic function $\lambda(v)$ as in Algorithm $stTree4$.

D. Network Coding Techniques

In this section, we use the network coding techniques in order to achieve further improvement in terms of minimizing congestion and reducing the number of overflows. The network coding technique enables, under certain conditions, to share edges that belong to different nets. For example, in the graph depicted in Figure 47(c) there are two minimum Steiner trees, one transmitting signal a from source s_1 and the second transmitting signal b from source s_2. These two trees clash at the middle edge (emanating from t_1), resulting in an overflow. This conflict can be resolved by coding at node t_1, which effectively allows two trees to share certain edges. Similarly, our algorithm will identify pairs of nets that share terminals and then apply network coding techniques to reduce overflow.

1. Previous Work on Network Coding

The problem of routing of multiple nets with shared terminals is related to the problem of establishing efficient multicast connections in communication networks. The network coding technique was proposed in a seminal chapter by Ahlswede et al. [1]. It was shown in [1] and [17] that the capacity of the multicast networks, i.e., the number of packets that can be sent simultaneously from the source node s to all terminals is equal to the minimum size of a cut that separates s from each terminal. Li et. al. [17] proved that *linear network codes* are sufficient for achieving the capacity of the network. In a subsequent work, Koetter and Médard [10] developed an algebraic framework for network coding. This framework was used by Ho et al. [11] to show that linear network codes can be efficiently constructed through a randomized algorithm. Jaggi et al. [12] proposed a deterministic polynomial-time algorithm for finding feasible network codes in multicast networks. An initial study of applicability of network

coding for improving the routability of VLSI designs appears in [71]. In [72] Gulati et al. used network coding for improving the routability of FPGAs, focusing on finding nets that are suitable for network coding. To the best of our knowledge, this is the first work that proposes efficient algorithms for finding the congestion-aware network coding topologies for VLSI circuits.

2. Network Coding Algorithm

We proceed to present the algorithm we use for constructing congestion-aware coding networks that reduce congestion and overflow.

The algorithm includes the following steps. First, we identify the subset S' of S that includes nets that go through edges with overflow. Second, we identify pairs of nets in S' that share at least three common terminals. Next, we check, for each such pair of nets (n_i, n_j) whether we can replace the Steiner trees for n_i and n_j by a more efficient routing topology with respect to congestion and overflow.

More specifically, let (n_i, n_j) be a pair of nets in S' that share at least three terminals. Let s_i and s_j be the source nodes of these nets. We denote the set of terminals shared by n_i and n_j by T_{ij}. We also denote by T'_i the set of terminals in T_i that do not belong to T_{ij}, i.e., $T'_i = T_i \setminus T_{ij}$. Similarly, we denote by T'_j the set of terminals in T_j that do not belong to T_{ij}, i.e., $T'_j = T_j \setminus T_{ij}$. Next, we find two congestion-aware Steiner trees ϕ_i and ϕ_j that connect s_i to T'_i and s_j to T'_j. These trees can be identified by one of the algorithms presented in Section C. The parameter $\eta(e)$ for each $e \in G$ is updated after finding ϕ_i and ϕ_j.

Finally, we find a congestion-aware network coding topology $\hat{\phi}$ that connects s_i and s_j to the common set of terminals T_{ij} in an iterative manner. First, we let $\hat{\phi}$ to be a Steiner tree with source s_i and terminals T_{ij}. All edges of $\hat{\phi}$ are always assigned zero cost. We then sort the terminals T_{ij} in the increasing order of their distance

(in the original graph) from s_j and process them in that order. For each terminal $t \in T_{ij}$, we reverse all the edges in the path $P_{i,t}$ between source s_i and terminal t and find a shortest path $P_{j,t}$ between source s_j and terminal t. Then, for each link $e(v,u) \in P_{j,t}$ we perform the following procedure. If there exists a link $e'(u,v)$ in $\hat{\phi}$, we remove $e'(u,v)$ from $\hat{\phi}$, otherwise, we add $e(v,u)$ to $\hat{\phi}$. A sample execution of this procedure is shown in Figure 49. It is easy to verify that the algorithm produces a feasible network coding topology, i.e., a topology that ensures that for each terminal $t \in T_{ij}$ there are two edge disjoint paths that connect s_i and s_j with t. The formal description of algorithm for identifying the network coding topology, referred to as Algorithm *NC*, is given in Figure 48.

After the execution of the algorithm, we determine whether the total cost of $\hat{\phi}$, ϕ_i, and ϕ_j is less than the total cost of the original Steiner trees for nets n_i and n_j. If there is a reduction in terms of cost, the two original Steiner trees are replaced by $\hat{\phi}$, ϕ_i, and ϕ_j.

Our experimental results presented in Section E, show that the number of coding opportunities is relatively small. However, by applying the network coding technique on a limited number of nets we can achieve a significant reduction in the number of overflows. Also, since the network coding technique is applied to a limited number of nets, the overhead in terms of the number of additional required gates is relatively small.

E. Performance Evaluation

We have evaluated the performance of our algorithms using the ISPD98 routing benchmarks [73]. All the experiments are performed on a 3.2 GHz Intel Xeon dual-core

Algorithm *NC* $(G, s_i, s_j, T_{ij}, T'_i, T'_j)$

Find a Steiner tree ϕ_i connects s_i to terminals in T'_i
Find a Steiner tree ϕ_j connects s_j to terminals in T'_j
For each link $e \in T'_i \cup T'_j$ update $\eta(e)$
Find a Steiner tree $\hat{\phi}$ that connects s_i to terminals in T_{ij}
Assign zero cost to all edges in $\hat{\phi}$
For each terminal $t \in T_{ij}$ let $d(t)$ be the shortest distance between s_j and t
For all terminals $t \in T_{ij}$ in the increasing order of $d(t)$ do:
    Let $G'(V', E')$ be a graph formed from $G(V, E)$ by reversing all edges in $P_{i,t}$,
        where $P_{i,t}$ is a path from s_i to t in $\hat{\phi}$
    Find shortest path $P_{j,t}$ from source s_j to terminal t in $G'(V', E')$
    Assign zero cost to all edges of $P_{j,t}$
For each edge $e(v, u) \in P_{j,t}$ **do**
    If there exists an edge $e'(u, v) \in \hat{\phi}$, remove $e'(u, v)$ from $\hat{\phi}$
    Otherwise, add $e(v, u)$ to $\hat{\phi}$
Return $\hat{\phi}$, ϕ_i, and ϕ_j

Fig. 48. Algorithm *NC*

machine. In all experiments, we first run the Steiner tree tool *Flute* [74] in order to determine the initial routing of all nets in the benchmark. Next, we perform an iterative procedure, referred to as Phase 1, which processes each net with overflows and checks whether an alternative Steiner tree of lower cost and with smaller number of overflows exists and if yes, rips up the existing tree and replaces it with an alternative one. This phase uses one of the algorithms described in Section C. Phase 1 terminates when four subsequent iterations yield the same cost and the number of overflows, indicating that further reduction in the number overflows is unlikely.

Next, we check whether the application of the network coding technique can further reduce the number of overflows. This phase is referred to as the Phase 2. We first identify pairs of nets that have overflowed edges and share at least there terminals. We then apply Algorithm *NC*, presented in Section D, to find an alternative network

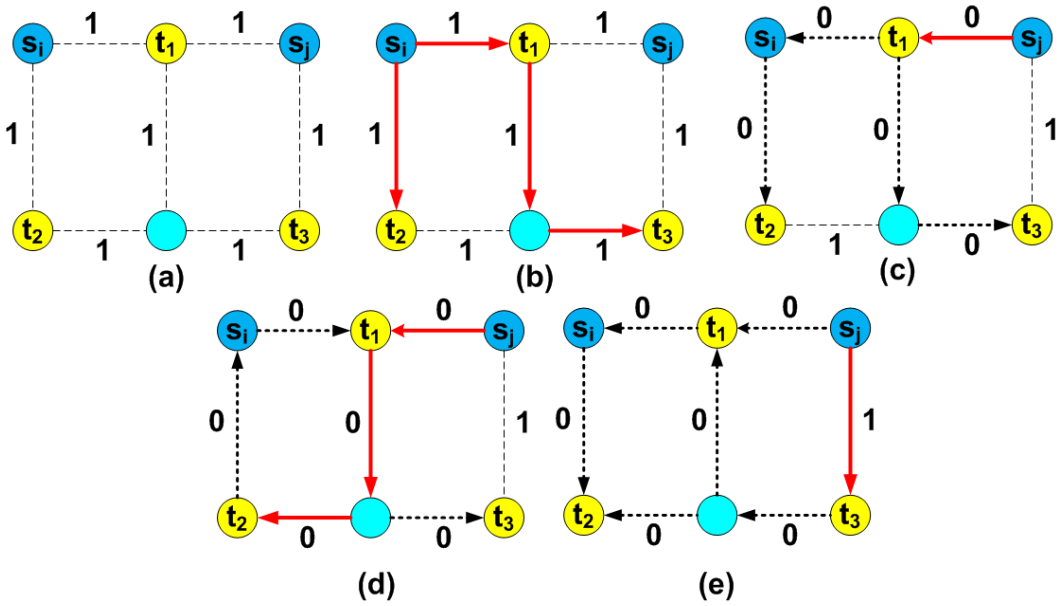

Fig. 49. Steps for finding network coding topology using Algorithm *NC*. (a) A graph G with two nets n_i and n_j that connect nodes s_i and s_j to terminals $T_1 = T_2 = T_{12} = \{t_1, t_2, t_3\}$. (b) A Steiner tree $\hat{\phi}$ connecting s_j to T_2. (c) Modified graph in which the costs of all edges found in $\hat{\phi}$ are set equal to zero and the edge connecting s_i to t_1 is reversed. A shortest path from s_j to t_1 is shown. (d) Modified graph in which the edges connecting s_i are t_2 are reversed. A shortest path from s_j to t_2 is shown. (e) Modified graph in which the edges connecting s_i to t_3 are reversed. A shortest path from s_j to t_3 is shown.

coding topology and perform rip-up and reroute if such a topology is beneficial in terms of reducing congestion and reducing overflows.

The experimental results are shown in Figures 50, 51, 52. Figures present average performance over all ten benchmarks. The cost function for this set of experiments was set according to Equation (8.1) with $\alpha \geq 10$. We have observed that larger values of α yield fewer overflows, but result in larger running times and increased wirelength. We also note that for Phase 1, Algorithm $stTree3$ shows the best performance in terms of reducing the total number of overflows as well as reducing the maximum overflow. In fact, Algorithm $stTree3$ eliminates all overflows in all benchmarks, except for ibm4 as given in Table V. We also note that Algorithms $stTree4$ and $stTree5$ yield Steiner trees with a smaller total wirelength. This is due to the fact the intelligent search algorithms favor paths that have small hop count.

We observe that the network coding technique results in a considerable reduction of the total number of overflows as well as reduces the maximum overflow. Furthermore, for each pair of nets for which we perform network coding, the number of required gates is small. Moreover, in all cases that we have encountered the network coding operations can be performed over finite field $GF(2)$, i.e., each encoding node can be implemented with a single XOR gate. Such gates incur minimum overhead because they can serve as buffers for long wires. An example of how network coding can help in reducing the wirelength of two nets in the ibm1 benchmark is shown in Figure 54.

Figure 53 compares the running times of the different Algorithms for Phase 1 and Algorithm NC for Phase 2. As expected Algorithm $stTree4$ is one of the fastest algorithms, whereas Algorithm $stTree1$ and $stTree5$ have running times comparable to Algorithm $stTree2$. Moreover, Algorithms $stTree4$ and $stTree5$ are faster than their counterparts (Algorithms $stTree1$ and $stTree2$, respectively). This is due to the

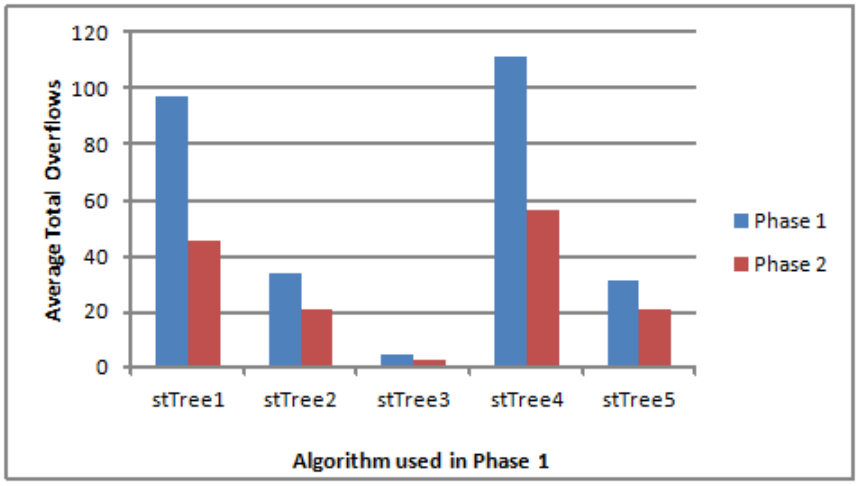

Fig. 50. Comparison of the average total overflow for five different algorithms for Phase 1 and Algorithm *NC* for Phase 2 on ISPD98 benchmark files using the polynomial cost function. Results present average over 10 ISPD98 benchmark files.

fact that intelligent search methods speed up the search by preferring nodes closer to the destination.

In the second set of experiments we evaluated the performance of three cost functions mentioned in Section 2 on the ISPD98 benchmarks using Algorithm $stTree3$ for Phase 1 and Algorithm *NC* for Phase 2. For cost function given by Equation (8.1) we used $\alpha \geq 10$, whereas for cost function given by Equation (8.2) we used $\beta \geq 50$ and for cost function given by Equation (8.3), $\rho(e)$ was a polynomial cost function with $\alpha \geq 10$. The results are shown in Table VI. Polynomial cost function showed

Table V. Performance Evaluation Using Algorithm $stTree3$ for Phase 1 and Algorithm NC for Phase 2 on ISPD98 Benchmarks with Polynomial Cost Functions.

	Phase 1			Phase 2		
	ov_{tot}	$ov_{\max}$	$wlen$	ov_{tot}	$ov_{\max}$	$wlen$
ibm1	14	1	81058	6	1	80727
ibm2	0	0	216299	0	0	216299
ibm3	0	0	188391	0	0	188391
ibm4	125	2	196106	93	2	195949
ibm5	0	0	415681	0	0	415681
ibm6	0	0	336105	0	0	336105
ibm7	0	0	466381	0	0	466381
ibm8	0	0	477404	0	0	477404
ibm9	0	0	508661	0	0	508661
ibm10	0	0	711201	0	0	711201

Table VI. Performance Evaluation of Different Cost Functions (using Algorithm $stTree3$ for Phase 1 and Algorithm NC for Phase 2) on ISPD98 Benchmarks.

	Polynomial Cost		Exponential Cost		History Based Cost	
	ov_{tot}	$ov_{\max}$	ov_{tot}	$ov_{\max}$	ov_{tot}	$ov_{\max}$
ibm1	0	0	117	1	0	0
ibm2	0	0	79	2	0	0
ibm3	0	0	0	0	0	0
ibm4	31	1	834	7	439	4
ibm5	0	0	0	0	0	0
ibm6	0	0	54	2	0	0
ibm7	0	0	82	1	0	0
ibm8	0	0	37	1	0	0
ibm9	0	0	15	1	0	0
ibm10	0	0	95	3	0	0

Table VII. Performance Evaluation Using Algorithm *stTree*3 for Phase 1 and Algorithm *NC* for Phase 2 on Modified ISPD98 Benchmarks (Decreased Both Horizontal and Vertical Capacity by 1 Unit) with Polynomial Cost Function.

	Flute			Phase 1			Phase 2			Number of XOR Gates
	ov_{tot}	$ov_{\max}$	$wlen$	ov_{tot}	$ov_{\max}$	$wlen$	ov_{tot}	$ov_{\max}$	$wlen$	
ibm1	3257	14	60209	362	4	78161	184	3	77231	5
ibm2	5678	22	166193	18	1	217673	4	1	217737	12
ibm3	2308	16	145777	1	1	183395	0	0	183391	2
ibm4	4833	15	162844	550	6	193832	440	6	193131	52
ibm5	5	4	410038	0	0	476251	0	0	473265	70
ibm6	5492	27	276012	3	1	334165	1	1	334163	0
ibm7	4665	15	363678	1	1	473743	0	0	473743	2
ibm8	5400	13	403502	3	1	476880	0	0	476864	0
ibm9	8545	14	411524	16	1	496563	9	1	496409	24
ibm10	7103	20	574743	4	1	713569	1	1	713553	18

Table VIII. Improvement Over MaizeRouter for Modified (Congested) IBM Benchmarks Using Algorithm *stTree*3 for Phase 1 and Algorithm *NC* for Phase 2, Using Polynomial Cost Function.

	Ver. Cap	Hor. Cap	MaizeRouter		Phase 1 + Phase 2	
			ov_{tot}	$ov_{\max}$	ov_{tot}	$ov_{\max}$
ibm1	11	13	43	1	34	1
ibm5	21	31	45326	25	45151	25
ibm8	17	28	649	9	641	9
ibm9	10	24	4006	9	3989	9

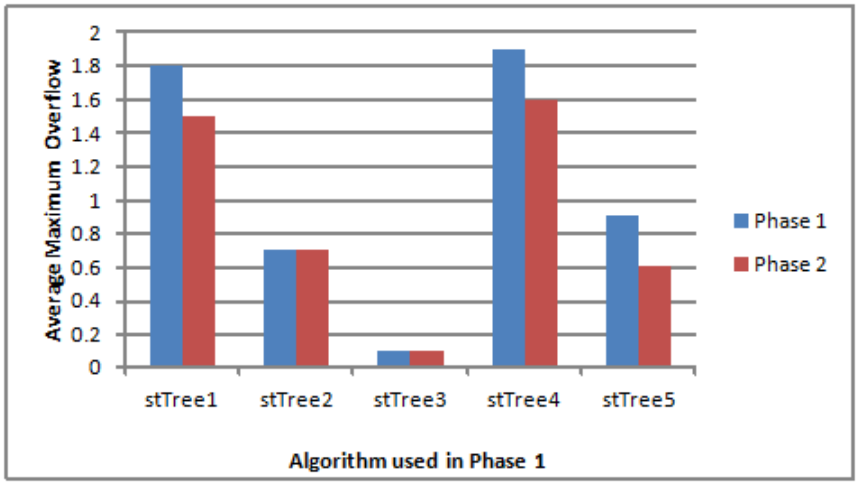

Fig. 51. Comparison of the average maximum overflow for five different algorithms for Phase 1 and Algorithm *NC* for Phase 2 on ISPD98 benchmark files using the polynomial cost function. Results present average over 10 ISPD98 benchmark files.

the best performance in terms of overflows.

In another set of experiments we worked on slightly modified ISPD98 benchmark files. The modification included reducing the vertical and horizontal capacity by one unit. For Phase 1 we used Algorithm *stTree*3 and then applied Algorithm *NC* in Phase 2. Polynomial cost function given by Equation (8.1) was used to check the performance on these more congested cases. The results are given in Table VII.

We have also conducted the same experiment on several selected benchmarks on the output of MaizeRouter [66]. In these experiments, we iteratively reduced the vertical and horizontal capacity of the ISPD98 benchmarks until we got overflows while running them through MaizeRouter. Then, we used the output of MaizeRouter as input to Phase 1 using Algorithm *stTree*3 and after that we have applied Algorithm *NC* in Phase 2. The cost function used was polynomial with $\alpha = 10$. The results are shown in Table VIII. The results demonstrate that Algorithms *stTree3*

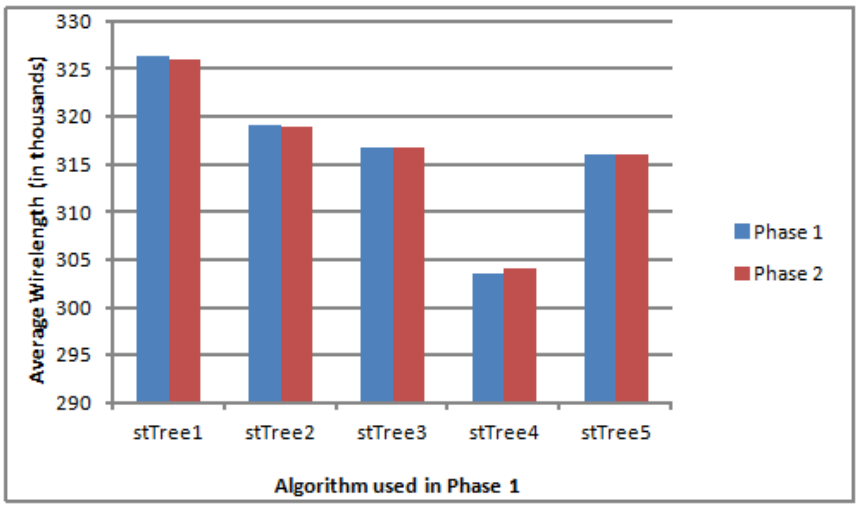

Fig. 52. Comparison of the average total wirelength for five different algorithms for Phase 1 and Algorithm *NC* for Phase 2 on ISPD98 benchmark files using the polynomial cost function. Results present average over 10 ISPD98 benchmark files.

combined with Algorithm *NC* perform well and can contribute to further reduction of the number of overflows.

F. Conclusions

In this chapter we presented several efficient techniques for rip-up-and-reroute stage of the global routing process. Our techniques are based on ripping-up nets that go through highly congested areas and rerouting them using efficient Steiner tree algorithms. We have considered several efficient Steiner tree algorithms as well as several cost functions that take into account congestion and overflow. We have also studied the application of the novel technique of network coding and showed that it can efficiently handle difficult routing cases, facilitating reduction in the number of overflows.

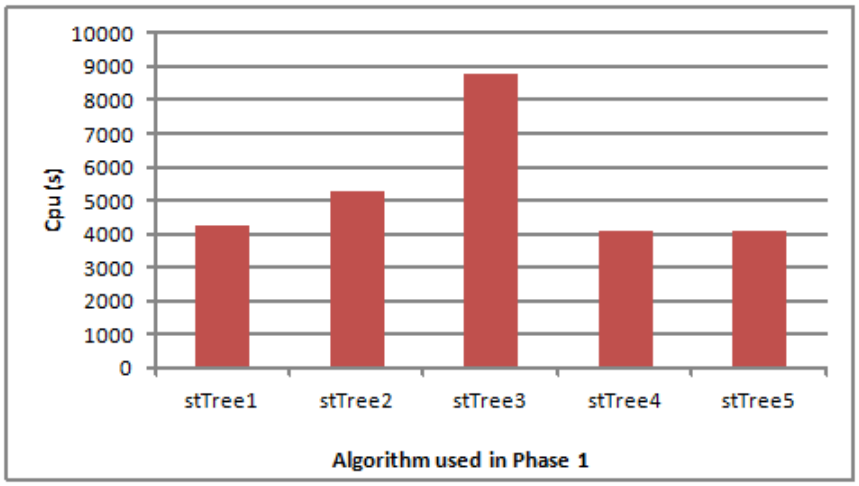

Fig. 53. Comparison of the average running time for five different algorithms for Phase 1 and Algorithm *NC* for Phase 2 on ISPD98 benchmark files using the polynomial cost function. Results present average over 10 ISPD98 benchmark files.

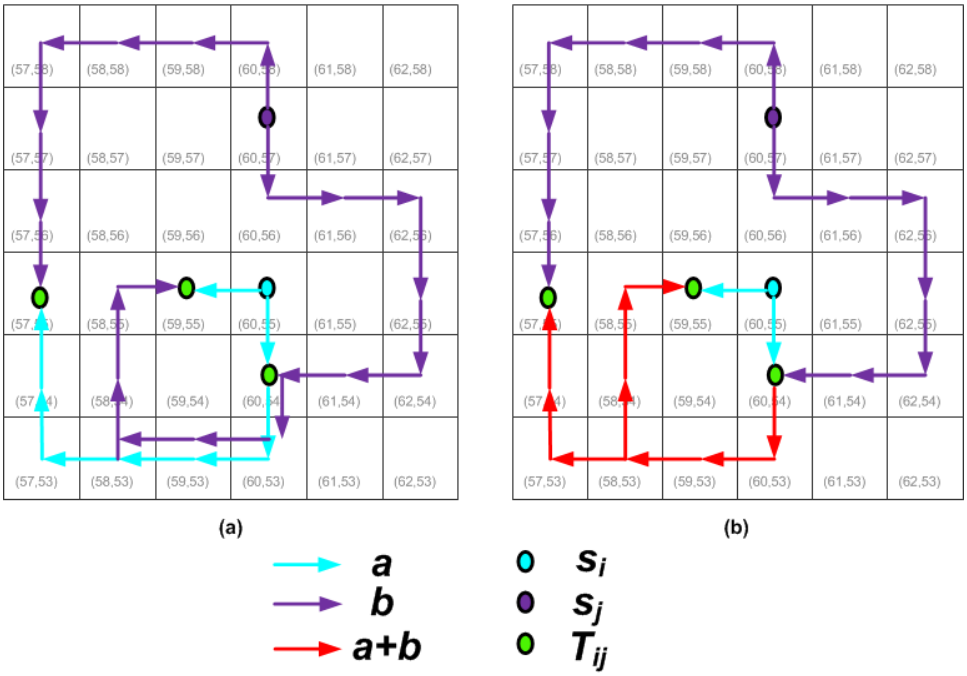

Fig. 54. A network coding topology for benchmark ibm1 for pair of nets $n_i = 66$ and $n_j = 1531$ computed using Algorithm *NC*. There are two nets n_i and n_j with sources $s_i = (60, 55)$, $s_j = (60, 57)$ and set of common terminals $T_{ij} = (59, 55), (60, 54), (57, 55)$. Part(a) shows routing layout without Network Coding. Part(b) shows routing layout with Network Coding. Example shows that network coding has helped to reduce congestion on edges $(60, 54)$-$(60, 53)$-$(59, 53)$-$(58, 53)$.

CHAPTER IX

CONCLUSION

The network coding has been shown to improve throughput, resilience and fault tolerance, and other quality of service parameters in communication networks as compared to the traditional techniques. The basic idea of the network coding techniques is to relish the "mixing" nature of the information packets, i.e., many algebraic operations can be performed over the data packets, whereas in traditional approaches information packets are treated as physical commodities (e.g., cars) over which algebraic operations (e.g., addition, subtraction etc) can not be performed, e.g., two cars can not be added together. The network coding has many applications including data communication, networking, and distributed storage. Since very first paper on the network coding in 2000 there has been a lot of work done on the network coding techniques, but many important questions are still unanswered. In this dissertation we try to answer some of the important open questions dealing with algorithms and application of the network coding. We study algorithms and applications of the network coding for dynamic networks, wireless networks, distributed storage, as well as other applications of the network coding technique. Our contributions can be divided into following four major parts.

In the first part of the dissertation we investigate the network coding for the dynamic networks, i.e., the network with frequently changing topologies and frequently changing sets of users. We focus on the network code design for dynamic networks. We first analyze the problem of maintaining the feasibility of a network code by minimizing the encoding coefficients that need to be modified (after a change in the network) to keep the network code feasible. Second, we present lower and upper bounds on the number of modifications required in case of an addition of a new

user or a failure of an edge. Third, we analyze the computational complexity of the problem in hand, and prove it to be NP-complete. Fourth, we present an algorithm for the network code design, and propose a new *path-based assignment* of encoding coefficients to construct a feasible network code. Fifth, we present a new method for assignment of encoding coefficients which is based on the prime numbers. The presented assignment scheme is distributed in nature, and does not require the full knowledge of the network topology. Sixth, we present an extensive simulation study on practical networks to show the advantage of the proposed schemes in practical scenarios.

In second part of the dissertation, we investigate the network coding problems in wireless networks. More specifically we focus on the *Index Coding* problem. The Index Coding problem has been proven to be NP-hard, and NP-hard to approximate. First, we propose an efficient exact, and several heuristic solutions for the Index Coding problem. Our numerical study suggests that the exact solutions can be efficiently identified for small instances, while the heuristic solutions with small computation time can achieve near optimal performance for larger instances. We then focus on finding approximate polynomial time solutions for the Index Coding problem with mathematically proven guarantees. Noting that the Index Coding problem has been proven to be NP-hard to approximate, we look at it from a different perspective and define the *Complementary Index Coding* problem. The goal of the Complementary Index Coding problem is to maximize the number of *saved transmissions*, i.e., the number of transmissions that are saved by employing encoding compared to the solution that does not involve coding. Second, we prove that the Complementary Index Coding problem can be approximated in several cases of practical importance. We investigate both the *multiple unicast* and *multiple multicast* scenarios for the Complementary Index Coding problem. In the multiple unicast scenario, each packet is

requested by a single client; while in the multiple multicast scenario, each packet can be requested by several clients. Third, we present approximation algorithms for finding *scalar* and *fractional* linear solutions for the multiple unicast scenario. Fourth, we show that for the multiple multicast scenario finding an approximation solution is NP-hard, and the multiple multicast scenario is NP-hard to approximate as well. Fifth, we focus on finding *sparse* solutions to the Index Coding problem with mathematically provable guarantee. In a sparse solution each transmitted packet is a linear combination of at most two original packets. We analyze both *scalar* and *fractional* versions of the problem, and provide polynomial time solution. For the scalar case, we present a polynomial time algorithm that achieves an approximation ratio of $2 - \frac{1}{\sqrt{n}}$. For the fractional case, we present a polynomial time algorithm that provides the optimal solution to the problem. Sixth, we perform extensive experimental study which demonstrate that our algorithms achieve good performance in practical scenarios.

In third part of the dissertation we consider the *Distributed Data Retrieval* problem i.e., the problem of accessing large data files stored at multiple locations across a content distribution, peer-to-peer, or massive storage network. The data can be stored in either original form, or encoded form at multiple network locations. We present a novel efficient polynomial-time solution for this problem that leverages the matroid theory. Experimental study on practical networks shows the advantage of our solution over alternative approaches.

In fourth part of the dissertation, we study the applications of the network coding for congestion mitigation and overflow avoidance in the global routing stage of Very Large Scale Integration (VLSI) physical design. We present novel techniques for reducing congestion and minimizing overflows. Our methods are based on ripping-up nets that go through the congested areas and replacing them with congestion-aware topologies. Our contributions can be summarized as follows. First, we present sev-

eral efficient algorithms for finding *congestion-aware* Steiner trees, i.e., trees that avoid congested areas of the chip. Second, we show that the novel technique of network coding can lead to further improvements in routability, reduction of congestion, and overflow avoidance. Thirdly, we present an algorithm for identifying efficient congestion-aware *network coding* topologies. We evaluate the performance of the proposed algorithms through extensive simulations using the International Symposium on Physical Design (ISPD) routing benchmarks.

REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204 –1216, Jul 2000.

[2] A. Dimakis, P. Godfrey, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," in *Proceedings of 26th IEEE International Conference on Computer Communications*, Anchorage , Alaska, May 2007, pp. 2000 –2008.

[3] M. Chaudhry, Z. Asad, A. Sprintson, and J. Hu, "Efficient rerouting algorithms for congestion mitigation," in *Proceedings of IEEE Computer Society Annual Symposium on VLSI*, Washington, DC, May 2009, pp. 43–48.

[4] Z. Bar-Yossef, Y. Birk, T. S. Jayram, and T. Kol, "Index coding with side information," in *Proceedings of 47th Annual IEEE Symposium on Foundations of Computer Science, 2006*, Berkeley, California, Oct. 2006, pp. 197–206.

[5] Y. Birk and T. Kol, "Informed-source coding-on-demand (iscod) over broadcast channels," in *Proceedings of Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies*, San Francisco, California, Mar.-Apr. 1998, pp. 1257–1264.

[6] E. Lubetzky and U. Stav, "Non-linear index coding outperforming the linear optimum," in *Proceedings of 48th Annual IEEE Symposium on Foundations of Computer Science*, Providence, Rhode Island, Oct. 2007, pp. 161–168.

[7] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *Proceedings of 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, Miami, Florida, Mar. 2005, pp. 2235–2245.

[8] M. Wang and B. Li, "Network coding in live peer-to-peer streaming," *IEEE Transactions on Multimedia*, vol. 9, no. 8, pp. 1554–1567, Dec. 2007.

[9] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "Xors in the air: Practical wireless network coding," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 497–510, Jun. 2008.

[10] R. Koetter and M. Medard, "An algebraic approach to network coding," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, Oct. 2003.

[11] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," in *Proceedings of IEEE International Symposium on Information Theory*, Pacifico Yokohama, Yokohama, Jun.-Jul. 2003, p. 442.

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

[13] T. Ho, M. Medard, and R. Koetter, "An information-theoretic view of network management," *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1295–1312, Apr. 2005.

[14] F. Zhao and M. Medard, "Online network coding for the dynamic multicast problem," in *Proceedings of IEEE International Symposium on Information Theory*, Seattle, Washington, Jul. 2006, pp. 1753–1757.

[15] T. Ho, B. Leong, M. Medard, R. Koetter, Y.-H. Chang, and M. Effros, "On the utility of network coding in dynamic environments," in *Processings of International Workshop on Wireless Ad-Hoc Networks*, Oulu, Finland, May.-Jun. 2004,

pp. 196–200.

[16] C. Fragouli, "An overview of network coding for dynamically changing networks," *International Journal of Autonomous and Adaptive Communications Systems*, vol. 2, no. 1, pp. 1–23, Mar. 2009.

[17] S.-Y. Li, R. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371 –381, Feb. 2003.

[18] M. Garey and D. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman and Company, San Francisco, California, 1979.

[19] S. Fortune, J. Hopcroft, and J. Wyllie, "The directed subgraph homeomorphism problem," *Theoretical Computer Science*, vol. 10, no. 2, pp. 111–121, Feb. 1980.

[20] T. Ho, "Networking from a network coding perspective," Ph.D. dissertation, MIT, May 2004.

[21] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum flow problem," in *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, Berkeley, California, May. 1986, pp. 136–146.

[22] "http://www.cs.washington.edu/research/networking/rocketfuel/," 2009.

[23] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "Xors in the air: practical wireless network coding," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, Pisa, Italy, Sep. 2006, pp. 243–254.

[24] B. Hassanabadi, L. Zhang, and S. Valaee, "Index coded repetition-based mac in vehicular ad-hoc networks," in *Proceedings of 6th IEEE Consumer Communications and Networking Conference*, Las Vegas, Nevada, Jan. 2009, pp. 1–6.

[25] S. Sorour and S. Valaee, "Adaptive network coded retransmission scheme for wireless multicast," in *Proceedings of IEEE International Symposium on Information Theory*, Seoul, Korea, 28 2009-july 3 2009, pp. 2577 –2581.

[26] M. Chaudhry and A. Sprintson, "Efficient algorithms for index coding," in *Proceedings of Annual IEEE International Conference on Computer Communications Workshops*, Phoenix, Arizona, Apr. 2008, pp. 1–4.

[27] Y. Wu, J. Padhye, R. Chandra, V. Padmanabhan, and P. Chou, "The local mixing problem," in *Proceedings of Information Theory and Applications Workshop*, San Diego, California, Feb. 2006.

[28] A. Blasiak, R. Kleinberg, and E. Lubetzky, "Index coding via linear programming," *Arxiv preprint arXiv:1004.1379*, vol. abs/1004.1379, 2010.

[29] N. Alon, E. Lubetzky, U. Stav, A. Weinstein, and A. Hassidim, "Broadcasting with side information," in *Proceedings of 49th Annual IEEE Symposium on Foundations of Computer Science*, Philadelphia, Pennsylvania, Oct. 2008, pp. 823–832.

[30] S. El Rouayheb, A. Sprintson, and C. Georghiades, "On the index coding problem and its relation to network coding and matroid theory," *IEEE Transactions on Information Theory*, vol. 56, no. 7, pp. 3187–3195, Jul. 2010.

[31] M. Langberg and A. Sprintson, "On the hardness of approximating the network

coding capacity," in *Proceedings of IEEE International Symposium on Information Theory*, Toronto, Ontario, Jul. 2008, pp. 315–319.

[32] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," in *Proceedings of the 38th annual Design Automation Conference*, Jun. 2001, pp. 530–535.

[33] N. Eén and N. Sörensson, "An extensible sat-solver," in *Proceedings of 6th International Conference on Theory and Applications of Satisfiability Testing*, May 2003, pp. 502–518.

[34] G. Tseitin, "On the complexity of derivation in propositional calculus," *Studies in constructive mathematics and mathematical logic*, vol. 2, pp. 115–125, 1970.

[35] S. El Rouayheb, M. Chaudhry, and A. Sprintson, "On the minimum number of transmissions in single-hop wireless coding networks," in *Proceedings of IEEE Information Theory Workshop*, Lake Tahoe, California, Sep. 2007, pp. 120–125.

[36] U. Feige and J. Kilian, "Zero knowledge and the chromatic number," in *Proceedings of Eleventh Annual IEEE Conference on Computational Complexity*, Philadelphia, Pennsylvania, May 1996, pp. 278–287.

[37] R. Duh and M. Fürer, "Approximation of k-set cover by semi-local optimization," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, ser. STOC '97, El Paso, Texas, United States, May 1997, pp. 256–264.

[38] R. Hassin and S. Lahav, "Maximizing the number of unused colors in the vertex coloring problem," *Information Processing Letters*, vol. 52, no. 2, pp. 87–90, 1994.

[39] V. V. Vazirani, *Approximation Algorithms*. Springer, 2004.

[40] R. Yuster and Z. Nutov, "Packing directed cycles efficiently," in *Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science*, Aug. 2004, pp. 1–15.

[41] M. Krivelevich, Z. Nutov, M. R. Salavatipour, J. V. Yuster, and R. Yuster, "Approximation algorithms and hardness results for cycle packing problems," *ACM Transaction on Algorithms*, vol. 3, no. 4, pp. 1–22, Nov. 2007.

[42] P. Seymour, "Packing directed circuits fractionally," *Combinatorica*, vol. 15, no. 2, pp. 281–288, 1995.

[43] D. Zuckerman, "Linear degree extractors and the inapproximability of max clique and chromatic number," in *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*.

[44] J. Suurballe and R. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, no. 2, pp. 325–336, 1984.

[45] S. Huang, A. Ramamoorthy, and M. Medard, "Minimum cost content distribution using network coding: Replication vs. coding at the source nodes," *Arxiv preprint arXiv:0910.2263*, 2009.

[46] A. Jiang, "Network coding for joint storage and transmission with minimum cost," in *Proceedings of 2006 IEEE International Symposium on Information Theory*, Seattle, Washington, july 2006, pp. 1359–1363.

[47] J. Edmonds, "Matroid intersection," *Annals of Discrete Mathematics*, vol. 4, pp. 39–49, 1979.

[48] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*. Springer Verlag, 2003.

[49] E. Lawler, *Combinatorial optimization: networks and matroids*. Dover Pubns, 2001.

[50] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Networks Flows*. Prentice-Hall, 1993.

[51] J. Oxley, *Matroid Theory*. Oxford University Press, 2006.

[52] R. Karp, E. Upfal, and A. Wigderson, "Constructing a perfect matching is in random nc," *Combinatorica*, vol. 6, no. 1, pp. 35–48, 1986.

[53] J. Lee and J. Ryan, "Matroid applications and algorithms," *INFORMS Journal on Computing*, vol. 4, no. 1, pp. 70–98, 1992.

[54] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring link weights using end-to-end measurements," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, Nov. 2002, pp. 231–236.

[55] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[56] N. Sherwani, *Algorithms for VLSI physical design automation*. Kluwer Academic Publishers, 1995.

[57] J. Hu and S. S. Sapatnekar, "A survey on multi-net global routing for integrated circuits," *Integration: the VLSI Journal*, vol. 31, no. 1, pp. 1–49, 2002.

[58] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Predictable routing," in *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*. San Jose, California: IEEE Press, Nov. 2000, pp. 110–114.

[59] C. Albrecht, "Provably good global routing by a new approximation algorithm for multicommodity flow," in *Proceedings of the 2000 international symposium on Physical design*, Apr. 2000, pp. 19–25.

[60] J. Cong, J. Fang, and Y. Zhang, "Multilevel approach to full-chip gridless routing," in *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, Nov. 2001, pp. 396–403.

[61] Y. W. Chang and S. P. Lin, "Mr: a new framework for multilevel full-chip routing," *IEEE Transactions on Computer-Aided Design*, vol. 23, no. 5, pp. 793–800, May 2004.

[62] M. Cho and D. Pan, "Boxrouter: a new global router based on box expansion and progressive ilp," in *Proceedings of 43rd ACM/IEEE Design Automation Conference*, San Francisco, California, Jul. 2006, pp. 373–378.

[63] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "Boxrouter 2.0: architecture and implementation of a hybrid and robust global router," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, San Jose, California, Nov. 2007, pp. 503–508.

[64] M. Pan and C. Chu, "Fastroute 2.0: A high-quality and efficient global router," in *Proceedings of Asia and South Pacific Design Automation Conference*, Pacifico Yokohama, Yokohama, Jan. 2007, pp. 250–255.

[65] J. Roy and I. Markov, "High-performance routing at the nanometer scale," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, San Jose, California, Nov. 2007, pp. 496–502.

[66] M. Moffitt, "Maizerouter: Engineering an effective global router," in *Proceedings*

of Asia and South Pacific Design Automation Conference, Seoul, Korea, Mar. 2008, pp. 226–231.

[67] G. Robins and A. Zelikovski, "Improved steiner tree approximation in graphs," in *Proceedings of 11th ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, California, Jan. 2000, pp. 770–779.

[68] S. Voß, "Steiner's problem in graphs: Heuristic methods," *Discrete Appl. Math.*, vol. 40, no. 1, pp. 45–72, 1992.

[69] M. P. D. Arag ao and R. F. Werneck, "On the implementation of mst-based heuristics for the steiner problem in graphs," in *Proceedings of 4th International Workshop on Algorithm Engineering and Experiments*, San Francisco, California, Jan. 2002, pp. 1–15.

[70] R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of a^*," *Journal of the ACM*, vol. 32, no. 3, pp. 505–536, 1985.

[71] N. Jayakumar, K. Gulati, S. Khatri, and A. Sprintson, "Network coding for routability improvement in vlsi," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, San Jose, California, Nov. 2006, pp. 820–823.

[72] K. Gulati and S. P. Khatri, "Improving FPGA routability using network coding," in *Proceedings of the 18th ACM Great Lakes symposium on VLSI*, Orlando, Florida, May 2008, pp. 147–150.

[73] C. J. Alpert, "The ISPD98 circuit benchmark suite," in *Proceedings of the 1998 international symposium on Physical design*, Monterey, California, Apr. 1998, pp. 80–85.

[74] C. Chu, "Flute: Fast lookup table based wirelength estimation technique," in *Proceddings of IEEE/ACM International Conference on Computer Aided Design*, San Jose, California, Nov. 2004, pp. 696–701.

VITA

Mohammad Asad Rehman Chaudhry received his B.Sc., and M.Sc., degrees both in Electrical Engineering from The University of Engineering and Technology, Lahore in 2001 and 2004 respectively. He received his Ph.D., in Electrical and Computer Engineering from Texas A&M University in 2011.

He may be reached at 214 Zachry Engineering Center, College Station, TX, U.S.A. 77843-3128. His email is masadch@gmail.com.