Evolutionary Approaches Toward Practical Network Coding

by

Minkyu Kim

B.S., Seoul National University (1998)
S.M., Massachusetts Institute of Technology (2003)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2008

Author .....
Department of Electrical Engineering and Computer Science
August 28, 2008

Certified by...
.....
Muriel Médard
Professor
Thesis Supervisor

Certified by.
.....
Una-May O'Reilly
Principal Research Scientist
Thesis Supervisor

Accepted by .....
.....
Terry P. Orlando
Chair, Department Committee on Graduate Students

Evolutionary Approaches Toward Practical Network Coding

by

Minkyu Kim

Submitted to the Department of Electrical Engineering and Computer Science
on August 28, 2008, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

There have been numerous studies showing various benefits of network coding. However, in order to have network coding widely deployed in real networks, it is also important to show that the amount of overhead incurred by network coding can be kept minimal and eventually be outweighed by the benefits network coding provides. Owing to the mathematical operations required, network coding necessarily incurs some additional cost such as computational overhead or transmission delay, and as a practical matter, the cost of special hardware and/or software for network coding. While most network coding solutions assume that the coding operations are performed at all nodes, it is often possible to achieve the network coding advantage for multicast by coding only at a subset of nodes. However, determining a minimal set of the nodes where coding is required is NP-hard, as is its close approximation; hence there are only a few existing approaches each with certain limitations.

In this thesis, we develop an evolutionary approach toward a practical multicast protocol that achieves the full benefit of network coding in terms of throughput, while performing coding operations only when required at as few nodes as possible. We show that our approach operates in a very efficient and practical manner such that it is distributed over the network both spatially and temporally, yielding a sufficiently good solution, which is at least as good as those obtained by existing centralized approaches but often turns out to be much superior in practice.

We broaden the application areas of our evolutionary approach by generalizing it in several ways. First, we show that a generalized version of our approach can effectively reveal the possible tradeoff between the costs of network coding and link usage, enabling more informed decisions on where to deploy network coding. Also, we demonstrate that our approach can be applied to investigate many important but, because of the lack of appropriate tools, largely unanswered questions arising in practical scenarios based on heterogeneous wireless ad hoc networks and fault-tolerant optical networks. Finally, further generalizing our evolutionary approach, we propose a novel network coding scheme for the general connection problem beyond multicast, for which no optimal network coding strategy is known. Our coding scheme allows general random linear coding over a large finite field, in which decoding is done only

at the receivers and the mixture of information at interior nodes is controlled by evolutionary mechanisms.

Thesis Supervisor: Muriel Médard
Title: Professor

Thesis Supervisor: Una-May O'Reilly
Title: Principal Research Scientist

Acknowledgments

I would like to thank my advisor Muriel Médard for her guidance and patience throughout my journey at MIT. I am greatly indebted to her for my development as a professional and also as a person. I would also like to thank my second advisor Una-May O'Reilly for her invaluable help and support. Without her encouragement this thesis would still be far from completion. I feel truly fortunate to have had an opportunity to be advised by these two gifted and caring individuals. I am also thankful to my thesis reader Michelle Effros for her insight and thorough comments which greatly improved this thesis.

I acknowledge that various parts of the work are from the papers on which I have collaborated with wonderful colleagues. I thank Chang Wook Ahn for introducing to me an interesting field of research; Varun Aggarwal for so many brilliant ideas; Wonsik Kim for his great help in simulations.

I would like to express my deepest gratitude to my parents whose faithful support and unfailing trust have been truly invaluable throughout my life. I also thank my sister and her husband for staying close as a family away from home. I am so grateful to my almost-a-year old daughter Kristin Dahyun for the joys and laughters she has brought into our lives. Lastly but foremost, I would like to thank my wife and best friend Kyungsun for her love, support and patience. Without her, my life would have been just incomplete.

This research was supported by the Korea Foundation for Advanced Studies; by the National Science Foundation under grant nos. CCR-0093349, CCR-0325496 and through University of Illinois subaward no. 2003-07092-1; by the Office of Naval Research under grant no. N00014-05-1-0197; by the Air Force Office of Scientific Research under grant no. FA9550-06-1-0155; by the United States Army under award no. W911NF-05-1-0246 through University of California subcontract no. S0176938; by the Defense Advanced Research Projects Agency under grant no. HR0011-08-1-0008.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

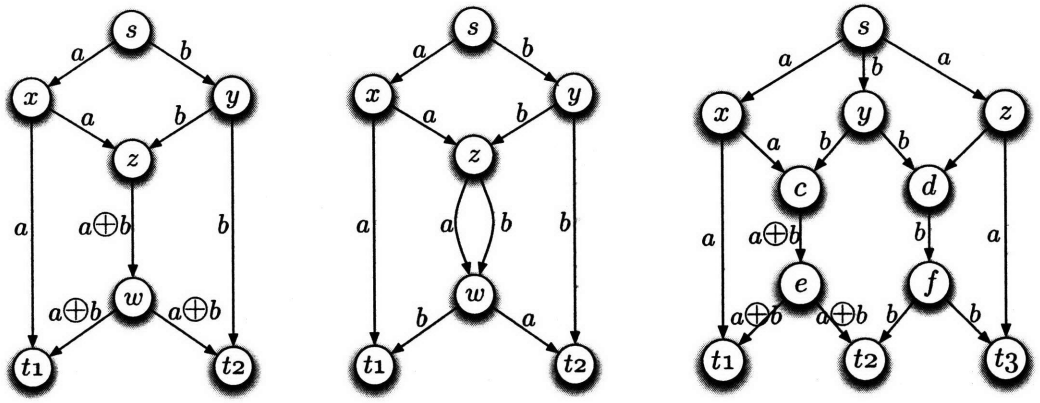
Network coding is a novel technique that generalizes routing. In traditional routing, each interior network node, which is not a source or receiver node, simply forwards the received data or sends out multiple copies of it. In contrast, network coding allows interior network nodes to perform arbitrary mathematical operations to combine the data received from different links. Network coding has been shown to offer numerous advantages over traditional routing such as optimal multicast throughput [1, 36, 38], minimum-cost multicast [42,58], improved security [25,28], and efficient network management [26], among many others.

However, owing to the mathematical operations required for coding at interior nodes, network coding necessarily incurs some additional cost, compared with traditional routing, such as computational overhead or transmission delay. Also, as a practical matter, each network node may need some special hardware and/or software to handle such network coding operations. Hence, network coding capability must be regarded as network resources that need to be well allocated, possibly in an optimized manner, to balance its cost and benefit.

While most network coding solutions assume that the coding operations are performed at all nodes, it is often possible to achieve the network coding advantage for multicast by coding only at a subset of nodes. Consider the following example.

Example 1 *In the canonical example of network A (Figure 1-1(a)) [1], where each*

link has a unit capacity and the desired multicast rate from s to t_1 and t_2 is 2, only node z needs to combine its two inputs while all other nodes perform routing only. If we suppose that link (z, w) in network A has capacity 2, which we represent by two parallel unit-capacity links in network B (Figure 1-1(b)), a multicast of rate 2 is possible without network coding. In network C (Figure 1-1(c)), where node s wishes to transmit data at rate 2 to the 3 leaf nodes, network coding is required at either node c or node d, but not both. □



(a) Network A (b) Network B (c) Network C

Figure 1-1: Sample networks for Example 1.

Given that performing network coding operations only at a subset of the nodes may be enough, a natural question to ask when deploying network coding for multicast is when and where network coding is required to obtain the desired benefit. For instance, if network coding is handled at the application layer, we can minimize the cost of network coding by identifying the nodes where access up to the application layer is not necessary. If network coding is integrated in the buffer management of a router, it is important to understand where and how many such special routers must be deployed.

However, as will be discussed more in detail in Chapter 2, determining a minimal set of the nodes where coding is required is NP-hard, and its close approximation is also NP-hard [34]. Given the hardness of the problem, there are only a few existing approaches, each with certain limitations, to the problem of minimizing the amount

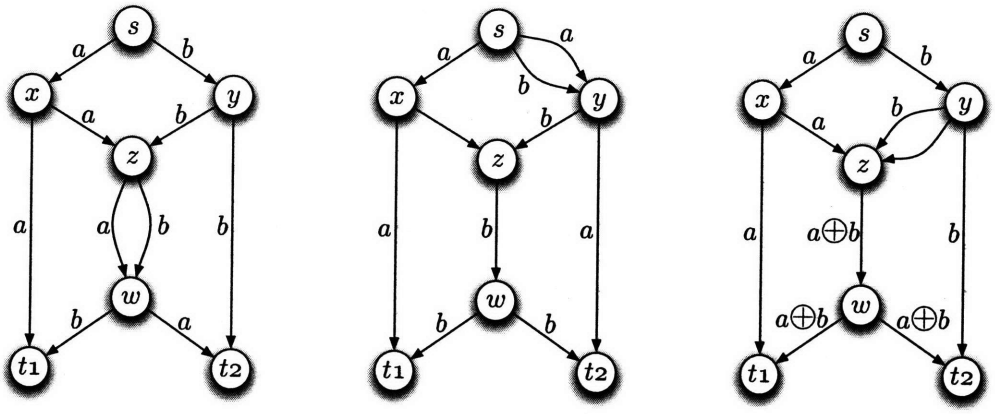
of resources engaged in network coding.

In Chapters 3 and 4, we develop an evolutionary approach to this network coding resource optimization problem, based on a Genetic Algorithm (GA) that operates on a set of candidate solutions which it improves sequentially via mechanisms inspired by biological evolution (e.g., recombination/mutation of genes and survival of the fittest). We show that our approach yields far superior solutions much more efficiently than other existing approaches.

Note that, in [42], network coding is shown to allow for minimum-cost multicast, which is an NP-complete task with traditional routing alone; i.e., network coding transforms a presumably intractable task to a polynomially solvable problem. However, if we take the cost of network coding into account as we mentioned earlier, what is achieved with the method in [42], in fact, is to have the cost of link usage minimized at the expense of a maximal cost of network coding.

Though it is necessary to *assume* network coding at all possible nodes initially to calculate a minimum-cost subgraph as described in [42], eventually there may not be many nodes in the resulting subgraph where network coding is actually required. If network coding is indeed necessary at some nodes in the found subgraph, one may wish to know whether it is because we constrained the information flow onto the selected minimum-cost subgraph, and if so, whether providing some extra capacity would then eliminate the requirement of network coding, i.e., whether there exists a tradeoff between the coding and link costs. Let us consider another example.

Example 2 *Suppose that we have three networks as depicted in Figure 1-2, in which each link has a unit capacity and a unit cost and the desired multicast rate from s to t_1 and t_2 is 2. In network B (Figure 1-2(a)), the target multicast rate is achievable without coding, incurring a total link cost of 10. To reduce the link cost to 9, we have to remove one of the two links between nodes z and w, making coding necessary at node z. In network B' (Figure 1-2(b)), by removing link (x, z), we can establish multicast connections of rate 2 without coding using only 9 links, whereas removing one of the two links between nodes s and y necessitates use of (x, z) and coding at node z in the remaining graph. In network B'' (Figure 1-2(c)), though one of the two*



(a) Network B (b) Network B' (c) Network B''

Figure 1-2: Sample networks for Example 2.

links between nodes y and z is redundant, coding at node z is necessary regardless of whether one of the redundant links is removed or not. □

As illustrated in Example 2, reducing link usage first by subgraph selection may give rise to the necessity of coding in the remaining subgraph, but we may also choose not to do coding while allowing some extra link cost. For some networks, such as network B, minimizing link usage first always increases the requirement for coding, whereas for some others, like network B', it depends on how a minimum-cost subgraph is chosen. Also, there are networks, e.g., network B'', where reducing the link cost first does not increase the coding cost. Hence, whether there exists such a tradeoff can be considered a topological property of the network.

If it is possible to identify the tradeoff in a given network, one may use the information to make decisions on the deployment of network coding such that coding happens only at the places where a significant amount of link cost is saved, or one may not want to employ network coding at all if the amount of the saved link cost turns out to be negligible for the topology. On the other hand, if the given network is known to have no such tradeoff, we can simply calculate a minimum-cost subgraph, based on which network coding resources can be optimized separately without sacrificing optimality.

In Chapter 5, we show that determining whether there exists such a tradeoff also

turns out to be NP-hard and thus we extend our evolutionary approach to investigate the issue of tradeoff between the coding and link costs. While having to deal with two objectives, i.e., coding and link costs, the extended approach maintains the key structure of the approach for coding resource optimization, hence the practical effectiveness can be carried over.

Now, if we generalize the networking scenario beyond multicast such that each receiver node may demand a different set of information out of multiple sessions, the problem of finding the optimal application of network coding still remains open. For instance, linear coding, which suffices to achieve the optimal transmission rate in the multicast case, is shown to be insufficient for optimal coding in the multi-session case [12]. Even within linear coding, in contrast with the multicast case for which the whole theory of network coding is well founded on Ahlswede et al.'s [1] coding theorem, there is no coding theorem to determine with bounded complexity whether the given set of communication demands can be satisfied with linear coding.

The difficulty in finding an optimal network coding strategy has constrained most research on inter-session network coding to focus on a number of significantly restricted classes of network coding in terms of the location of decoding and/or the type of coding operations allowed [15, 17, 23, 31, 55]. Moreover, if we wish to consider large problems or slightly relax the restrictions on the coding schemes, most existing approaches become difficult to implement in practice.

In Chapter 7, we investigate how our evolutionary approach can further be generalized to fill the wide gap between the presumably difficult quest for optimal inter-session coding and many existing restrictive, yet impractical coding schemes. More precisely, we present a novel randomized linear coding scheme, in which decoding happens at the receivers while interior nodes perform random linear coding with selective mixture of information, controlled by an evolutionary framework similar to that used for multicast.

1.1 Contributions

There have been numerous studies showing various benefits of network coding. However, in order to have network coding widely deployed in real networks, it is also important to show that the amount of overhead incurred by network coding can be kept minimal and eventually be outweighed by the benefits network coding provides.

In this thesis, we develop an evolutionary approach toward a practical multicast protocol that achieves the full benefit of network coding in terms of throughput, while performing coding operations only when required at as few nodes as possible. For the NP-hard problem of minimizing network coding resources for multicast, there are a few existing approaches, but each suffers from implementation challenges. Our approach is distributed over the network both spatially and temporally, yielding a sufficiently good solution, which is at least as good as those obtained by existing approaches but often turns out to be much superior in our simulations. Our approach leads to the first practical solution to the problem, making a distributed network coding protocol, combined with the distributed random network coding scheme, where the resources used for coding are optimized on the fly in the setup phase and afterwards coding operations are performed only at the found minimal set of coding nodes.

The application areas of our evolutionary approach can be extended beyond the problem of coding resource optimization. We show that our evolutionary approach, with a slight generalization, can effectively reveal the possible tradeoff between the costs of network coding and link usage, which can be considered a topological property of a given network, enabling more informed decisions on where to deploy network coding. We also demonstrate that our approach can be applied further to investigate many important but, due to the lack of appropriate tools, largely unanswered questions: e.g., how to integrate network coding in heterogeneous wireless networks where coding and non-coding legacy nodes coexist, and whether network coding would be useful for protection in optical networks where the savings due to network coding may be overshadowed by the high cost of data processing in the electronic domain.

Further generalizing our evolutionary approach, we propose a novel network coding strategy for the general connection problem beyond multicast, for which no optimal network coding strategy is known. Our coding strategy allows fairly general random linear coding over a large finite field, in which decoding is done only at the receivers and the mixture of information at interior nodes is controlled by evolutionary mechanisms. Through simulations, we demonstrate how our coding strategy surpasses existing end-to-end XOR coding schemes in terms of effectiveness and practicality.

Finally, another important contribution of this thesis is that it paves the way for further application of more advanced evolutionary algorithms to various aspects of network coding. There have been many recent developments in the field of evolutionary computation significantly improving the scalability of the traditional simple GA, on which our current approaches are based. A variety of such more recent evolutionary computing techniques have been successfully applied to many practical but extremely difficult problems that were difficult to handle with more traditional optimization methods. In this thesis, we focus on how a number of important network coding problems can be formulated into the evolutionary framework and also how utilizing the dependency structures implied by network topologies can significantly improve the performance of the algorithm, even using standard simple GA. There are numerous possibilities for further network coding strategies with far superior performance that combine the attributes investigated in this thesis with more recent advances in evolutionary computing.

1.2 Outline of Thesis

The rest of the thesis is organized as follows:

- Chapter 2 presents the network coding resource optimization problem as well as the main idea behind our evolutionary approach, with a brief introduction to GAs.

- Chapter 3 describes the details of our algorithm for the network coding resource

optimization problem, focusing mainly on the computational aspects of the algorithm.

- Chapter 4 shows how our algorithm can be implemented in a distributed manner over the network both in spatial and temporal domains.

- Chapter 5 generalizes our algorithm to deal with multiple objectives, revealing the tradeoff between the costs of network coding and link usage.

- Chapter 6 displays two more practical scenarios for which our algorithm is effectively utilized, answering many interesting and important questions that are difficult to tackle using prior techniques.

- Chapter 7, further generalizing our algorithm, proposes a novel network coding scheme, called selective random linear coding, for the generalized connection problem beyond multicast.

- Chapter 8 concludes the thesis with some directions for future work.

Chapter 2

Minimizing Network Coding Resources in Multicast

2.1 Problem Formulation

We begin our discussion with the standard framework introduced in [33], where we model the network by a directed multigraph $G = (V, E)$ with unit-capacity links. Connections with larger capacities are represented by multiple links. We assume that the given multigraph is acyclic as directed cycles can be easily avoided by selecting a subgraph, for instance, using the distributed algorithm in [24] or simple heuristics [48].

Later (in Chapter 6) we will show how our network model can be generalized to deal with more practical network applications in wireless networks with broadcast transmissions or optical networks with directed cycles.

We consider the single source multicast scenario in which a single source node $s \in V$ wishes to transmit data at a given rate R to a set $T \subset V$ of receiver nodes, where $|T| = d$. Rate R is said to be achievable if there exists a transmission scheme that enables all d receivers to receive all of the information sent. We consider only scalar linear coding, where a node's output on an outgoing link is a linear combination of the inputs from its incoming links. Scalar linear coding is sufficient for multicast [36].

We assume that the given target rate R is achievable when network coding is allowed at all nodes. Then, our objective is to determine a minimal set of nodes

where coding is required in order to achieve this rate. However, the network coding resource optimization problem is known to be difficult, as is it close approximation.

Theorem 1 *[34, Theorem 4] The problem of finding a minimal set of nodes where network coding is required to achieve the given multicast rate R is NP-hard. Moreover, it is also NP-hard to approximate the minimal number of coding nodes within any multiplicative factor or within an additive factor of $|V|^{1-\epsilon}$ for any constant $\epsilon > 0$.*

2.2 Related Work

Given the hardness of the problem, there are only a few existing approaches, each with certain limitations, to the coding resource optimization problem.

Fragouli et al. [18] show that coding is required at no more than $(d-1)$ nodes in acyclic networks with 2 unit-rate sources and d receivers. This result, however, is not easily generalized to more than 2 sources. They also present an algorithm to construct a minimal subtree graph. For target rate R, they first select a subgraph consisting of R link-disjoint paths to each of d receivers and then construct the corresponding labeled line graph in which they sequentially remove the links whose removal does not affect the achievable rate.

Langberg et al. [34] derive an upper bound on the number of required coding nodes for both acyclic and cyclic networks. They give an algorithm to construct a network code that achieves the bounds, where the network is first transformed such that each node has degree at most 3 and each of the links is sequentially examined and removed if the target rate is still achievable without it.

Let us now illustrate how these two approaches would apply to network B depicted in Fig. 1-1(b). In Fragouli et al.'s approach, either link l_1 or l_2 may be removed while selecting the subgraph, which renders coding at node z necessary in the remaining subgraph. If, on the other hand, both links l_1 and l_2 are retained in the subgraph, whether coding is required depends on the order in which the links are visited to construct a minimal subtree graph; for a randomly chosen order of link inspection, coding is required with probability $\frac{1}{2}$.

Langberg et al.'s method first decomposes nodes z and w as in Fig. 2-1; for this network, there are many sequences of link removals that result in a subgraph where coding is required: e.g., if l_1 is the first visited link, node z_4 must perform coding. Empirical tests show that the probability that coding is required for random link removals is about 0.68.

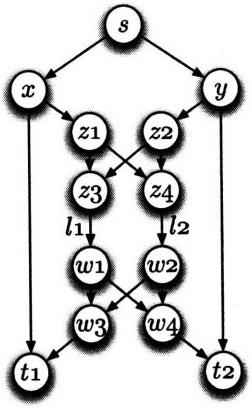


Figure 2-1: Decomposed network for Langberg et al.'s method.

Note that both of the above approaches remove links sequentially in a greedy fashion, assuming that network coding is done at all nodes with multiple incoming links in the remaining graph. Unless a good link traversal order is found, the quality of the solution may not be improved much.

Bhattad et al. [3] give linear programming formulations for the problems of optimizing over various resources used for network coding, based on a model allowing continuous flows. Their optimal formulations, however, involve a number of variables and constraints that grows exponentially with the number of receivers, which makes it hard to apply the formulations to the case of a large number of receivers, even at the price of sacrificed optimality.

2.3 Main Idea of Our Approach

Network coding is said to be necessary at a node if on at least one of its outgoing link the transmitted output must involve coding operations among multiple inputs to

achieve the given multicast rate. First of all, it is clear that no coding is required at a node with only a single incoming link because these nodes have nothing to combine with, of which formal proof appears in [58, Lemma 2]. Therefore, we only need to consider the nodes with multiple incoming links, which we refer to as *merging nodes*, to find if network coding is necessary.

In order to determine if network coding is necessary for each outgoing link of a merging node, we need to verify whether we can constrain the output on that link to depend only on a single input without destroying the achievability of the given multicast rate. To be precise, for a nonempty set L of links, network coding is not necessary on the links in L if there exists a feasible network code in which the data transmitted on each of the links in L depends only on the input from a single incoming link at the associated node.

It is shown in [33] that the achievability of the desired multicast rate is equivalent to whether the transfer matrices for the desired connections, which describe the relationship between the input from the source and the output to the receivers, are all nonsingular. By representing the determinants of those system matrices as polynomials, the task is equivalent to testing whether the product of the determinants, which we refer to as polynomial $\mathcal{P}$ for future reference, is nonzero over the ring of polynomials in variables of the coding coefficients [33]. Note that such coding coefficients that appear in polynomial $\mathcal{P}$ are the coefficients used for 1) encoding source data at the source, 2) linearly combining inputs to calculate outputs at interior nodes, and 3) decoding received data at the receivers.

Consider a merging node v with $d_{in}(\geq 2)$ incoming links and $d_{out}(\geq 1)$ outgoing links. For each pair of the $i \in \{1, ..., d_{in}\}$-th incoming link and the $j \in \{1, ..., d_{out}\}$-th outgoing link, polynomial $\mathcal{P}$ contains an associated coding coefficient, which signifies the coefficient multiplied by the input from the i-th incoming link when calculating the output on the j-th outgoing link.

Note that whether coding is necessary on the j-th outgoing link is equivalent to whether we can make all but one of the d_{in} associated coefficients zero without forcing polynomial $\mathcal{P}$ to be zero. Therefore, in order to determine whether coding is necessary

at node v, we need to keep track of whether each of the coefficients (there are a total of $d_{in}d_{out}$ such coefficients) is zero or not.

Hence, our problem is now transformed into a combinatorial optimization problem where we need to find the set of the binary variables that results in the smallest number of coding nodes while maintaining the achievability of the given multicast rate.

As illustrated in network C of Example 1, whether node v must code varies depending on which other nodes are coding. Thus, deciding whether or not to code potentially depends on all other nodes, involving a selection out of exponentially many possible choices. To address this scaling issue, we employ a GA-based search method.

In the next two chapters, we describe the details of our approach, based on the main idea presented here. Before proceeding, we provide a brief introduction to GAs.

2.4 A Brief Introduction to Genetic Algorithms

GAs are stochastic search methods that mimic genetic phenomena such as gene recombination, mutation and survival of the fittest. GAs have been applied to a large number of scientific and engineering problems, including many combinatorial optimization problems in networks (e.g., [16], [11]). The main control flow of the standard form of GA, called *simple GA*, is shown in Fig. 2-2 [47].

```
initialize population;
evaluate population;
while termination criterion not reached
  {
   select solutions for next population;
   perform crossover;
   perform mutation;
   evaluate population;
  }
```

Figure 2-2: Main control flow of simple GA.

Simple GA [47] operates on a set of candidate solutions, called a *population*. Each solution is typically represented by a bit string, called a *chromosome*. Each chromosome is assigned a *fitness value* that measures how well the chromosome solves the problem at hand, compared with other chromosomes in the population. From the current population, a new population is generated typically using three genetic operators: *selection*, *crossover* and *mutation*. Chromosomes for the new population are selected randomly (with replacement) in such a way that chromosomes that are more fit are selected with higher probability. For crossover, chromosomes are randomly paired, and then two chromosomes in each pair exchange a subset of their bit strings to create two offspring. Chromosomes are then subject to mutation, which refers to random flips of the bits applied individually to each of the new chromosomes. The process of evaluation, selection, crossover and mutation forms one *generation* in the execution of simple GA. The above process is iterated with the newly generated population successively replacing the current one. Simple GA terminates when a certain stopping criterion is reached, e.g., after a predefined number of generations.

There are several aspects of our problem suggesting that a GA-based method may be a promising candidate: GAs have proven to work well if the space to be searched is large, but known not to be perfectly smooth or unimodal, or even if the space is not well understood [47] (which makes traditional optimization methods difficult to apply). Note that the search space of our problem is apparently not smooth or unimodal with respect to the number of coding links and the structure of the space consisting of the feasible binary vectors is not well understood. Since the problem is NP-hard, it is not critical that the calculated solution may not be a global optimum. Note also that, while it is hard to characterize the structure of the search space, once provided with a solution we can verify its feasibility and count the number of coding links therein in polynomial time. Thus, if the use of genetic operations can suitably limit the size of the space to be actually searched, a solution can be obtained fairly efficiently.

For the past few decades, a number of new GA frameworks have been proposed to enhance the scalability of simple GA. It has been pointed out that in simple

GA, traditional crossover operators, such as one-point or uniform crossover, may tend to disrupt correlated chunks of solutions and thus may fail to ensure effective juxtaposition of promising partial solutions [49]. Hence, the focus of more recent GA research has been to design a new set of genetic operators that do not disrupt important partial solutions yet ensure effective mixing of them. A common theme shared by those more recent GAs is to find and exploit the dependencies, often called *linkage*, among building blocks of the solution. In order to do so, new chromosome representations and genetic operations are developed using some probabilistic models, e.g., Bayesian networks [49].

Note, however, that the purpose of evolutionary algorithms including GAs in general is to serve as a stochastic search method that does not rely on any specific structure of the problem, i.e., a "black box" optimization method that can be applied to as wide a variety of problems with little known structures as possible. Given a specific problem, however, the desired properties of an optimization method can be very different since it is often beneficial to utilize as much of the given problem's structure as possible.

The most notable characteristic of our problem is that the linkage information is already implied to some extent by the network topology, as will be discussed in subsequent chapters. Hence, rather than employing those more sophisticated GA frameworks, we choose to focus on the simple GA framework and investigate effective utilization of the structural properties given by the network topology. Note that any structure found using simple GA may also be utilized later in more advanced GA frameworks for further improved scalability.

Chapter 3

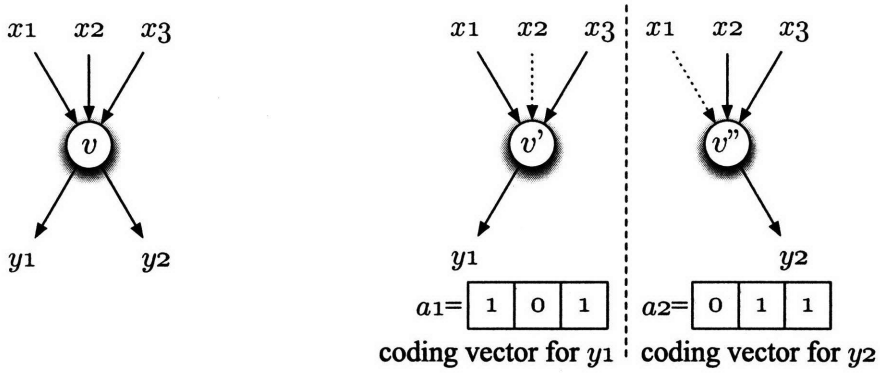
Computational Aspects

In this chapter, we describe the details of our approach for the network coding resource optimization problem, focusing mainly on its computational aspects, i.e., how our network coding problem is mapped into the GA framework and how each component of GA is designed. The actual implementation of our approach can be either centralized, assuming all information is available at a single location, or distributed over the network nodes as we will show in the next chapter. In either case, the computational part will remain the same as described in this chapter.

3.1 Chromosome Representation

Let us begin with describing how the chromosomes (i.e., candidate solutions) are composed in our GA framework. For each merging node with $d_{in}(\geq 2)$ incoming links and $d_{out})(\geq 1)$ outgoing links, we assign a binary variable a_{ij} to each pair of the $i \in \{1, ..., d_{in}\}$-th incoming link and the $j \in \{1, ..., d_{out}\}$-th outgoing link, which is 1 if the input from incoming link i contributes to the linearly coded output on outgoing link j and 0 otherwise. Hence, there is a one-to-one correspondence between the binary variables we introduce here and the network coding coefficients at the merging nodes in the algebraic network coding framework [33] mentioned in Section 2.3.

For the j-th ($j = 1, ..., d_{out}$) outgoing link, we refer to the set of associated binary variables $a_j = (a_{ij})_{i \in \{1,...,d_{in}\}}$ as a *coding vector* (see Fig. 3-1 for an example).



(a) Merging node v

(b) Coding vectors for outgoing links

Figure 3-1: Node v with 3 incoming and 2 outgoing links is associated with two coding vectors $a_1 = (a_{11}, a_{21}, a_{31})$ and $a_2 = (a_{12}, a_{22}, a_{32})$.

Each chromosome is then simply the collection of all those coding vectors. Hence, once a chromosome is given, it indicates which inputs will contribute to which outputs at each of the merging nodes. If we denote by $d_{in}(v)$ and $d_{out}(v)$ the in-degree and out-degree of node v, then the length of a chromosome is

$$m = \sum_{v \in \mathcal{V}} d_{in}(v) d_{out}(v), \tag{3.1}$$

where $\mathcal{V}$ is the set of all merging nodes.

3.2 Fitness Function

Let us now consider how we define the fitness value of each chromosome (i.e., relative strength of each candidate solution). Note that, given a chromosome, the number of coding *links* can be easily calculated by counting the number of coding vectors with at least two 1's. For the number of *nodes* where coding is required, we count the number of nodes that involve at least one coding link.

It remains to verify whether the desired multicast rate is still achievable when we constrain the transmission as specified by the given chromosome, i.e., the connections corresponding to the variables having 0 should not be used. Let us refer to a chromosome as *feasible* if the desired rate is still achievable with the constraint and

infeasible otherwise. Then, we define the fitness value F of chromosome $\underline{z}$ as follows:

$$F(\underline{z}) = \begin{cases} \text{number of coding nodes,} & \text{if } \underline{z} \text{ is feasible,} \\ \infty, & \text{if } \underline{z} \text{ is infeasible.} \end{cases} \tag{3.2}$$

Though the number of coding *nodes* is a more appropriate resource to minimize in practice, we may alternatively consider the number of coding *links* because essentially network coding happens when a node places output data onto its outgoing links. Thus the number of coding links may serve as an estimate of the amount of computational overhead incurred by network coding. In such a case, we only need to change the value of $F(\underline{z})$ in (3.2) to the number of coding links for feasible chromosomes.

To test the feasibility of the chromosome, we have to decide whether polynomial $\mathcal{P}$, defined in Section 2.3, is nonzero after zeroing out all the coding coefficients associated with 0 in the given chromosome. For this polynomial identity test, there exist numerous algorithms either randomized (most famously by Schwartz and Zippel [51] and more recently [7,35]) or deterministic ([29,39]).

In particular, we find that Ho et al.'s random linear coding approach [27] can be very useful, though in a slightly different context: whereas in [27] it serves as a method to construct a valid network code that achieves the given rate, here we use it to decide the achievability of the given rate. Specifically, we let each interior node transmit onto each outgoing link a random linear combination of the inputs excluding those associated with 0 in the chromosome, i.e., the coding coefficients corresponding to 0 are assigned deterministic zero in the finite field. Then, each receiver node can determine the feasibility by performing Gaussian elimination.

Note that this feasibility test entails a bounded error in declaring the chromosome infeasible, which corresponds to the coding error in [27] with the upper bound $1 - (1 - d/q)^{\nu}$ where q is the size of the finite field used for coding and ν is the maximum number of links in any set of links constituting a flow solution from the source to any receiver. However, there is no error in declaring the chromosome feasible. Thus, a feasible chromosome may mistakenly be declared infeasible but not vice versa. Moreover, we can lower the error bound as much as we desire at an additional cost

of computation; i.e., we may increase the size q of the finite field or perform multiple tests for each chromosome, declaring the chromosome feasible if it is found to be feasible at least once.

There are two notable advantages of this feasibility test method. First, the feasibility test can be done in a distributed manner over the network. Second, if the solution is feasible, a useable network code is obtained as a byproduct of the feasibility test. Both of these properties will turn out to be very useful, as can be seen in the next chapter.

Alternatively, solely for the sake of an efficient simulation, one may use the randomized test developed originally by Schwartz, Zippel, and many others (e.g., [51]). If we assign random integers from a finite set S and operate in the real field, the randomized test, which now has the error probability no greater than $(1 - d\nu/|S|)$, can run substantially faster than that performing matrix computations in a large finite field. Note, however, that due to the roundoff errors, one may have to use a numerical method such as the condition number, which signifies that the matrix is singular. The condition number is efficiently calculated by singular value decomposition and is considered a numerically reliable indicator of matrix singularity [10, 19].

3.3 Iteration of Algorithm

With the chromosomes and fitness function as defined above, our approach can be summarized into the main control flow shown in Fig. 3-2, which is based on the standard form of simple GA [47]. We describe the details of each procedure in the order of occurrence.

Population Initialization [C1]

The initial population is typically constructed randomly such that each component of the chromosomes is assigned 0 or 1 with equal probabilities. Note, however, that the size of the population, typically not exceeding a few hundred, is usually much smaller than the size of the entire search space, and thus it is very unlikely that a feasible

```
[C1] initialize population;
[C2] evaluate population;
[C3] while termination criterion not reached
     {
[C4]   select solutions for next population;
[C5]   perform crossover;
[C6]   perform mutation;
[C7]   evaluate population;
     }
[C8] perform greedy sweep;
```

Figure 3-2: Main control flow of our approach to the network coding resource optimization problem.

chromosome happens to be included in a randomly generated initial population. As a result, the algorithm may fail to yield a single feasible solution for a considerable number of early generations.

Thus, for the given population size N, we randomly generate $N-1$ chromosomes randomly and add a chromosome with all 1's that signifies the case that coding is performed at all merging nodes. Note that this all-one chromosome is feasible by assumption but has the worst fitness value among the feasible chromosomes. In our preliminary experiments, the insertion of an all-one chromosome improves the performance of the algorithm very significantly. For instance, without an added all-one chromosome, the algorithm almost always ended with the population of only infeasible chromosomes even for a mid-sized problem.

Initial Fitness Evaluation [C2]

The fitness values of the initial chromosomes are calculated using random linear coding as described in Section 3.2.

Termination Criterion [C3]

The iteration continues until the generation number reaches the predefined limit. It may be useful to adopt an additional termination criterion such that the iteration is terminated if no progress is made in the best fitness value of the population for

another predefined number of generations.

Selection [C4]

Based on the calculated fitness values, we generate a new population for the next generation by performing *tournament selection* [47] as follows. We form a tournament by randomly selecting M chromosomes from the current population, out of which only the best one is selected (with replacement) into the new population. The size M of the tournament, also called the *selection pressure*, is a predetermined parameter. We repeat this random tournament N times, where N is the population size.

Crossover [C5]

Chromosomes selected for the new population are randomly paired and undergo *uniform crossover* [47], where each pair of chromosomes is selected for crossover with a given probability (*mixing ratio*) and the two chromosomes in a selected pair exchange each bit with another given probability (*crossover probability*).

Mutation [C6]

Then we perform *binary mutation* [47] on each chromosome in the new population. In binary mutation, each bit in each chromosome is flipped independently with a given probability (*mutation rate*).

Fitness Evaluation [C7]

After all genetic operations, crossover and mutation, are performed, the new population is evaluated as in Section 3.2 and replaces the older one with the exception, called *elitism*, that the best chromosome in the older chromosome survives intact while the worst chromosome in the new population is dropped.

Greedy Sweep [C8]

This is a novel operator we introduce for our problem. For greedy sweep, we inspect the best chromosome obtained at the end of the iteration and switch each of the remaining 1's to 0 if it can be done without violating feasibility. Note that this additional procedure can only improve the solution. Since this is a one-time operation, we may repeat the feasibility test as many times as desired when verifying whether each remaining 1 can be flipped. Moreover, as will be discussed in the next section, the greedy sweep operator provides a theoretical performance bound that is as good as the best known bound by Langberg et al. [34].

3.4 Performance Bound

Let us denote by $\underline{z}$ the best chromosome found after the iteration of the algorithm including the greedy sweep operation. Then, with an arbitrarily high probability, $\underline{z}$ gives the same upper bound as in [34, Theorem 5] on the number of coding links, which again is the upper bound on the number of coding nodes. To be precise, suppose that for the greedy sweep operation, we repeat the feasibility test k times for each remaining 1. Let l be the length of the chromosome and ν be the maximum number of links in any set of links constituting a flow solution from the source to any receiver.

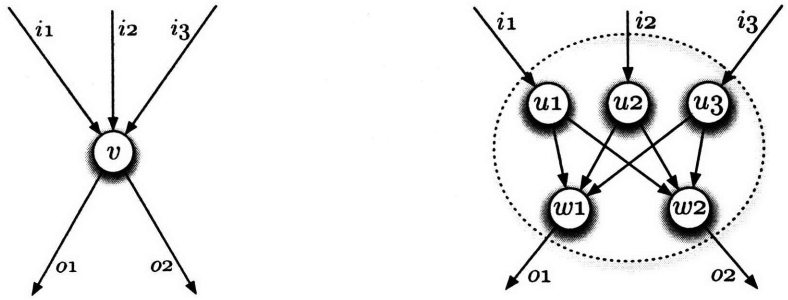
Theorem 2 *The number of coding links associated with $\underline{z}$ is upper bounded by R^3d^2 with bounded probability p such that*

$$p \geq (1-(1-(1-\frac{d}{q})^{\nu})^{k})^{l}. \tag{3.3}$$

Proof: Recall that the error in the algebraic feasibility test described in Section 3.2 is one-sided; i.e., a feasible solution can be mistakenly declared infeasible, but not vice versa. When we verify whether we can flip each remaining 1, an error occurs only if

all k feasibility tests are erroneous whereas the currently considered binary variable can actually be flipped. Hence, the probability that each decision is correct during the greedy sweep operation is at least $1 - (1 - (1 - d/q)^{\nu})^k$. Since each decision is made independently and there are at most l decisions to be made, the probability p that all decisions in the greedy sweep operation are correct is lower bounded as in (3.3).

Let us now decompose each merging node as follows to interpret algebraic network coding in the graph theoretic framework. Consider a node v with d_{in} incoming links $(i_1, ..., i_{d_{in}})$ and d_{out} outgoing links $(o_1, ..., o_{d_{out}})$. Let us first decompose the node by introducing d_{in} *incoming auxiliary nodes* $(u_1, ..., u_{d_{in}})$ and redirect node v's d_{in} incoming links to each of the incoming auxiliary nodes (see Fig. 3-3). Similarly, we create d_{out} *outgoing auxiliary nodes* $(w_1, ..., w_{d_{out}})$ and let node v's d_{out} outgoing links be the only outgoing link of each of the outgoing auxiliary nodes. We then introduce a link between each pair of incoming and outgoing auxiliary nodes.



(a) Before decomposition (b) After decomposition

Figure 3-3: Decomposition of a node with $d_{in} = 3$ and $d_{out} = 2$.

Note that for any network code on the original graph, we can consider the same network code on the decomposed graph such that each node u_j $(j = 1, ..., d_{in})$ simply forwards the input from link i_j and each node w_k $(k = 1, ..., d_{out})$ transmits the same linear combination as the output on link o_k in the original graph.

Then, there is a one-to-one correspondence between the set of binary variables in the chromosome and the set of the introduced links between auxiliary nodes. It is clear

that our algebraic feasibility test, assuming that its error probability is negligible, is equivalent to 1) deleting first all the introduced links between auxiliary nodes associated with 0 in the chromosome and 2) then calculating the max-flows between the source and the receivers.

Switching 1 to 0 in a chromosome implies deleting the corresponding link in the decomposed graph. Hence, in the decomposed graph associated with $\underline{z}$, there should be no link between auxiliary nodes that can be removed without violating the achievability. One can easily verify that there is a one-to-one correspondence between the links between auxiliary nodes in our decomposed graph and the set of the *paths within* the gadget Γ_v introduced in [34] (see Fig. 2 in [34]). Now we can replace all non-merging nodes with a degree larger than 3 by the gadgets and greedily remove links, which, however, is irrelevant of the number of coding links. By doing so, from $\underline{z}$ we can construct a simple instance, as defined in [34], and it gives the desired upper bounds on the number of coding links [34, Lemma 14]. □

Note that the probability bound (3.3) can be made arbitrarily close to 1 by increasing k. More specifically, given a target probability x, the value of k that makes the bound (3.3) exceed x is given by

$$k = \lceil \log_{1-(1-d/q)^{\nu}}(1 - x^{1/l}) \rceil, \tag{3.4}$$

which can be kept fairly low, provided that the field size q is appropriately chosen so that the original bound on the probability of decision error (i.e., $1-(1-d/q)^{\nu}$, which is the probability of coding error in actual multicast) is not too high. For instance, if the original bound on the probability of decision error is 0.01, k is just 4 for target probability $x = 0.999$ and chromosome length l up to 100,000.

The above theorem provides a performance bound implying that our algorithm with greedy sweep performs at least as well as the algorithm in [34], which is a very useful property since such a performance bound is not often available for an evolutionary algorithm.

It is important to note that any feasible chromosome that did not go through even a single generation of our GA iteration would give the same bound after the greedy sweep operator. Thus, one may well suspect that the above performance bound may often be very loose, which has been verified through our preliminary experiments on some sample networks. The practical usefulness of the above performance bound will be discussed more later in Section 3.7.

3.5 Vector-Wise Representation and Operators

For a coding vector of length k, we may allow all possible 2^k strings, which we refer to as *bit-wise representation*[1].

Note, however, that once a coding vector has at least two 1's, replacing all the remaining 0's with 1 has no effect on whether coding is done and that substituting 0 with 1, as opposed to substituting 1 with 0, does not hurt the feasibility. Therefore, for a feasible chromosome, any coding vector with two or more 1's can be treated the same as the coding vector with all 1's.

Thus we may replace all vectors with two or more 1's by a single vector signifying *coded* transmission ("111...1"). In addition, we use k coding vectors for *uncoded* transmissions of the input received from one of the k incoming links ("100...0", "010...0", "001...0", ..., "000...1") and one coding vector indicating *no* transmission ("000...0"). With this representation, which we refer to as *vector-wise representation*, a length-k coding vector is allowed to take only $(k+2)$ strings. If we let w be the total number of coding vectors and k_i denote the length of the i-th vector $(i = 1, ..., w)$, the search space size is significantly reduced to $\prod_{i=1}^{w}(k_i+2)$, from $2^{\sum_{i=1}^{w} k_i}$ in the case of bit-wise representation.

Let us refer to the genetic operators, crossover and mutation, defined above in Section 3.3 as *bit-wise genetic operators*. In order to preserve the structure of the vector-wise representation, we need to define a set of new genetic operators, which

[1]In the GA community, the method for representing a candidate solution as a bit string is called *genotype encoding*; we avoid the use of this term to minimize confusion with the term encoding in the context of network coding.

we refer to as *vector-wise genetic operators*, as follows:

- **Vector-wise uniform crossover:** We let two chromosomes subject to crossover exchange each full coding vector, rather than each individual bit, independently with the given crossover probability.

- **Vector-wise mutation:** For each chromosome, we randomly select each coding vector independently with the given mutation probability and let each of the selected coding vectors take another string chosen uniformly at random out of the remaining strings allowed for its length.

Note that the vector-wise genetic operators can be interpreted as the standard genetic operators as described in Section 3.3 with a variable-length alphabet such that each coding vector of length k is represented as a single element from the alphabet of size $(k+2)$.

It is interesting to note that the benefit of the smaller search space size in the vector-wise representation in fact comes at the price of losing the information on the coding vectors whose bit-wise representation has two or more 1's and a positive number of 0's, which may serve as intermediate steps toward eventually an uncoded transmission. Also, whereas the *average* number of bits flipped by vector-wise mutation of a length-k coding vector for mutation rate α is $\frac{4k^2}{(k+1)(k+2)}\alpha$, which is smaller than that by the bit-wise mutation $(k\alpha)$, the probability that 2 or more bits are flipped is often much larger for vector-wise mutation; this may negatively affect the GA's ability to improve the solution through fine random changes. Hence, the overall effect of vector-wise representation and operators on the algorithm's performance is not straightforward to predict theoretically. An experimental evaluation of this question is done in the next section.

3.6 Guidelines for Parameter Selection

As is typical for a GA, our approach involves many parameters that affect its performance to varying extents. Though there is no way to find an "optimal" combination

of such parameters, there are some empirical guidelines that seem to work well in practice.

First, the size of the population often serves as an important factor for the ability of a GA to find a good solution [22]. If the population is too small, it is not likely that the GA would give a good solution. Increasing the population size generally enhances the quality of the solution a GA produces, but it may cost an excessively large amount of computational resources. Reference [22] provides an equation for the size of population required to find a guaranteed optimal solution within a give confidence level, based on a number of idealized assumptions. However, the given population size scales exponentially with the size of the problem and moreover, the given equation is pessimistic in the sense that it does not take into account the effects of more efficient genetic operators, for instance, that exploit the modularity of variables as in our case. In practice, an appropriate population size is often found by a method based on successive bisection of the given initial interval [49]. In this thesis, however, we just use a moderate population size of 200, unless noted otherwise, throughout the simulations in all chapters.

Also, the size of the tournament for selection needs to be chosen carefully. If the tournament size is too big, the algorithm may converge prematurely to lower quality solutions, and if it is too small, the search process may become inefficient, taking too much time exploring unimportant solutions. In our experiments, we set the tournament size to 10, which is 5% of the population size and within a reasonable range typically used in GA applications.

For other parameters, such as mixing ratio, crossover probability and mutation rate, there are certain ranges of values found to work well with many different problems [47], and thus we pick the values within the ranges and fix them throughout our simulations.

Table 3.1 summarizes the GA parameters used in most of simulations subsequently.

Table 3.1: GA parameters used in the most experiments.

Population size	200
Tournament size	10
Mixing ratio	0.8
Crossover probability	0.2
Mutation rate	0.02
Maximum generations	1000

3.7 Performance Evaluation

In Chapters from 3 to 5, we set up simulations on synthetic network topologies with known, possibly restrictive, structures to evaluate and compare the effects of different components of the algorithm. For these simulations, our objective is to minimize the number of coding *links* because it represents more accurately the amount of computational overhead incurred by network coding as mentioned in Section 3.2.

In Chapter 6, we present more realistic network applications and perform simulations based on ad hoc wireless networks and optical networks, in which cases we use the number of coding *nodes* as our objective value to minimize.

We mainly compare the effects of different chromosome representations and genetic operators, i.e., bit-wise vs. vector-wise. To demonstrate the effectiveness of the greedy sweep operator, we also show the solutions by our algorithm without greedy sweep. Then, for comparison with existing approaches, we performed experiments using the approaches by Fragouli et al. [18] ("Minimal 1") and Langberg et al. [34] ("Minimal 2"), assuming in both of the algorithms that link removal is done in a random order. For Minimal 1, the subgraph is also selected first by a greedy approach, which sequentially removes from the original graph any links whose removal does not destroy the achievability.

3.7.1 Experimental Setup

The bit-wise and vector-wise representations/genetic operators differ in both the size of search space and the way the genetic operators are applied. While evaluating the effect of the search space size reduction, we also want to investigate whether the ex-

ploitation of block level modularity by the vector-wise operators gives any significant improvement in the algorithm's performance. We thus set up two experiments: Experiment I compares the effect of the two chromosome representations combined with associated operators on the performance of the algorithm, while Experiment II tests the effect of the operators alone by isolating the effect of the different chromosome representations that lead to different space sizes.

Experiment I: We use two acyclic random networks, R-50 and R-75, generated by the algorithm in [44], with the detailed parameters given in Table 3.2. Note that the vector-wise representation reduces the size of the search space by 30.3 and 115 orders of magnitude for networks R-50 and R-75, respectively, compared with that in the case of the bit-wise representation. This experiment tests which pair of chromosome representation and genetic operators is better given the tradeoff in the search space size and ease of traversing the fitness landscape.

Experiment II: We construct a set of synthetic networks with only coding vectors of length 2. Note that for a coding vector of length 2, the two representations yield the same search space size ($2^k = k + 2$ when $k = 2$), but the vector-wise genetic operators retain their modularity. These networks are constructed by cascading a number of copies of network B used in Example 1 (Fig. 1-1(b)) such that the source of each subsequent copy of B is replaced by an earlier copy's receiver. We use fixed-depth binary trees containing 3, 7, 15, and 31 copies of B (henceforth called B-3, B-7, B-15 and B-31, respectively). For example, network B-7 is depicted in Fig. 3-4. Parameters of these networks are given in Table 3.2.

The advantage of using this type of network is that all of these networks have the known value 0 as the minimum number of coding nodes (i.e., multicast rate 2 is achievable without coding) and also that we can scale up the network size to investigate the payoff one obtains with modular operators as the search space size increases.

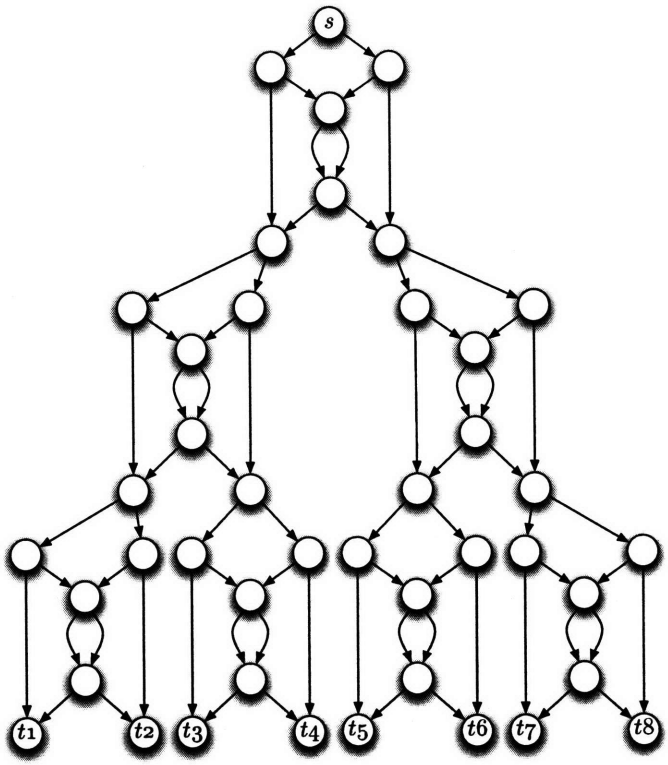


Figure 3-4: Network B-7 used in Experiment II

3.7.2 Experimental Results

Results for both experiments are summarized in Tables 3.3 and 3.4. The tables show the calculated minimum number of coding links, averaged over 30 runs, by each algorithm. The values in brackets are the standard deviations. The statistical significance of the difference between the cases of bit-wise and vector-wise representation and operators is measured by conducting paired t-tests and the p-values are reported in the last row of the table.

Experiment I: When coupled with greedy sweep, our algorithm outperforms the two existing approaches regardless of the representations and operators for both networks R-50 and R-75. Without greedy sweep, however, the performance of our algorithm with the bit-wise representation and operators deteriorates significantly for network R-75, yielding even worse results than the two existing approaches. In contrast, in the vector-wise case the performance remains the same for both networks. Between the bit-wise and vector-wise representations and operators, the vector-wise

Table 3.2: Details of the networks used in the experiments.

Network	Chromosome length	No. of coding vectors	Avg. len. of coding vectors	Search space size ($\log_{10}$) (Bit-wise/Vector-wise)
R-50	280	71	3.94	84.29/53.93
R-75	761	130	5.85	229.08/113.47
B-3	32	16	2	9.63/9.63
B-7	80	40	2	24.08/24.08
B-15	176	88	2	52.98/52.98
B-31	368	184	2	110.78/110.78

Table 3.3: Calculated minimum number of coding links, averaged over 30 runs, by different algorithms in Experiment I.

	R-50	R-75
Bit-wise	3.03(1.00)	5.97(1.25)
(w/o greedy sweep)	3.03(1.00)	38.30(2.56)
Vector-wise	2.17(0.38)	3.33(0.55)
(w/o greedy sweep)	2.17(0.38)	3.33(0.55)
Minimal 1	4.90(1.37)	9.50(2.16)
Minimal 2	4.33(1.37)	7.90(1.71)
p-value	$7.75e^{-5}$	$3.69e^{-13}$

case gives rise to a substantial performance gain over the bit-wise case with the statistical significance confirmed by the tabulated p-values.

Experiment II: Again our algorithm with the vector-wise representation and operators outperforms that with the bit-wise counterpart on average for all networks, while either of the two cases performs significantly better than the two existing approaches. This time, the greedy sweep operator has almost no influence on the performance of the algorithm. For networks B-3 and B-7, the vector-wise case finds the known optimal solution (0 coding links) in all of the 30 runs, while for networks B-15 and B-31, it succeeds to find the optimal solution 25 and 8 times, respectively. On the other hand, the bit-wise case does not find the optimal number of coding links in any of the 30 runs for networks B-7, B-15 and B-31.

Table 3.4: Calculated minimum number of coding links, averaged over 30 runs, by different algorithms in Experiment II.

	B-3	B-7	B-15	B-31
Bit-wise	0.70(0.65)	1.97(1.10)	4.93(1.34)	11.70(2.17)
(w/o greedy sweep)	0.70(0.65)	1.97(1.10)	4.94(1.34)	11.70(2.17)
Vector-wise	0.00(0.00)	0.00(0.00)	0.10(0.31)	0.90(0.76)
(w/o greedy sweep)	0.00(0.00)	0.00(0.00)	0.10(0.31)	0.90(0.76)
Minimal 1	3.00(0.00)	7.00(0.00)	15.50(0.00)	31.00(0.00)
Minimal 2	2.13(0.86)	4.37(1.25)	9.90(1.65)	19.97(2.66)
p-value	$2.17e^{-6}$	$1.01e^{-10}$	$3.40e^{-19}$	$8.73e^{-25}$

3.7.3 Discussions

Bit-wise vs. Vector-wise

Experiment I clearly indicates that the pair of vector-wise representation and operators is better than the bit-wise counterpart for the networks considered. We can thus conclude that the benefits of the smaller search space trump the challenge of the more difficult fitness landscape. For network R-50, the vector-wise case improves over the bit-wise case on average by a single coding link. Though small, this difference is statistically significant. For network R-75, without greedy sweep, the average difference in performance between the two cases is much higher, i.e., 34 coding links. This large difference in performance can be attributed to two specific factors: the much larger search space size (see Table 3.2) and larger average coding vector size. The difference also indicates that the information on the intermediate solutions that the bit-wise representation provides may not be particularly useful without guaranteeing that those intermediate steps ultimately lead to an uncoded transmission state.

Modularity or Exploratory Power

Experiment II demonstrates the superiority, by a remarkably large margin, of the vector-wise operators over the bit-wise operators. This prompts a further analysis of the difference between the two operators. When applied to the pair of coding vectors "00" and "11", the vector-wise crossover cannot result in either "01" or "10". However, for the bit-wise crossover, the pair of coding vectors "00" and "11" may

result in "00", "01", "10", or "11". It can be shown that with probability $\frac{1}{4}$ the two crossovers behave differently, if the population has equal frequency of all coding vector types. Let us recall that the vector-wise mutation leads to a smaller number of changed bits on average than the bit-wise mutation (Section 3.5). Nevertheless, the vector-wise mutation exhibits higher "exploratory power" than the bit-wise mutation in the sense that it is more likely to lead to changes in multiple bits. For the vector-wise mutation, given any coding vector, the remaining three coding vectors are equally likely to occur on mutation. Thus, if mutation rate is α, the probabilities of 0, 1, 2-bit change are $1-\alpha$, $\frac{2}{3}\alpha$, $\frac{1}{3}\alpha$, respectively, whereas those probabilities in the bit-wise case are $(1-\alpha)^2$, $2\alpha(1-\alpha)$, α^2, respectively. Provided that $\alpha < \frac{1}{3}$, the probability of 2-bit change is larger for the vector-wise mutation. A similar analysis can be done for the whole chromosome as well.

One may speculate that the better performance of the vector-wise operators is due to the higher exploratory power of the vector-wise mutation rather than the modularity of the operators. To confirm that it is the modularity of the operators that leads to the superior performance, we consider a new set of operators, called the *Matched Hamming Distance* (MHD) operators, where the MHD mutation leads to the statistically same Hamming distance changes as the vector-wise mutation, but exhibits no positional bias as to where the mutation is applied, and the MHD crossover is the same as the bit-wise crossover, neither of which imposes modularity. From Table 3.5 compared with Table 3, we observe that the MHD operators perform similarly to the bit-wise operators, but far worse than the vector-wise operators. This experiment increases our confidence that the respect for modularity enforced by the vector-wise operators is the main cause of the superior performance of the vector-wise operators.

Table 3.5: Calculated minimum number of coding links by our algorithm with the MHD genetic operators. Refer to Table 3.4 for comparison with the bit-wise or vector-wise operators.

	B-3	B-7	B-15	B-31
MHD	0.67(0.66)	2.13(1.04)	5.33(1.63)	12.03(2.79)

Effectiveness of Greedy Sweep

In Experiment I, our algorithm with the bit-wise representation and operators performs much worse than the two existing approaches. Such poor performance may be due to the fact that some parameters, such as the population size or the maximum number of generations, are not suitably chosen for the size of the search space. Even with such misadjusted parameters, if combined with greedy sweep, our algorithm performs much better than the two existing approaches. Hence, the greedy sweep operator may be useful as a safeguard that prevents the algorithm's poor performance due to misadjusted parameters.

Note, however, that with the block-wise representation/operations, our algorithm always produces solutions that are far better than the two existing approaches, and moreover, good enough so that further improvement by greedy sweep is never observed.

Hence, we may conclude that once the modularity of the variables is properly exploited by the block-wise representation and operators, the positive effect of the greedy sweep operator barely exists.

Chapter 4

Distributed Implementation

In this chapter, we present a distributed framework for our evolutionary approach described in the previous chapter. In particular, we show that our algorithm can be distributed in two ways, along spatial and temporal axes. Given that the main advantage of network coding based multicast is that both the subgraph optimization and the network code construction can be done in a distributed fashion [27, 42], the motivation for distributing the algorithm's operation spatially over the network nodes becomes apparent. The motivation for the second, temporal, axis of distribution is to maximize the efficient use of the computational resources in the network by pipelining successive sets of candidate solutions through the network, from source to receivers and back.

The most important benefit of this distributed implementation is that it enables a network coding protocol where the resources used for coding are optimized on the fly. In addition, we will show that the distribution reduces the computational complexity of the feasibility test and enhances the efficiency of the algorithm in terms of the time to convergence.

Note that, to the best of our knowledge, this is the first work that considers a doubly distributed GA structure. In contrast, a conventional distributed GA distributes the (sub)populations over multiple servers for parallel processing, but each chromosome is not processed in a distributed manner. A GA with the proposed two novel methods of distribution can be readily applied to a variety of other optimization

scenarios arising in communication networks (e.g., routing, resource allocation, etc.) or other connected systems where local decision variables are to be specified for the optimal performance of the whole system.

4.1 Spatial Distribution

Recall that in Section 3.2 we discussed how the feasibility test of a single chromosome can be done by employing random linear coding at interior nodes. Note that in doing so, each merging node only references the relevant portion of the chromosome, i.e., the coding vectors that indicate the operations at that node. Hence, we can divide up the population by letting each node handle only the coding vectors it needs from every chromosome in the population (see Fig. 4-1). Also, as will be discussed below, all genetic operations can be done independently at local interior nodes with some coordination information embedded in data packets. Thus, the whole population can be managed in a distributed manner over the network.

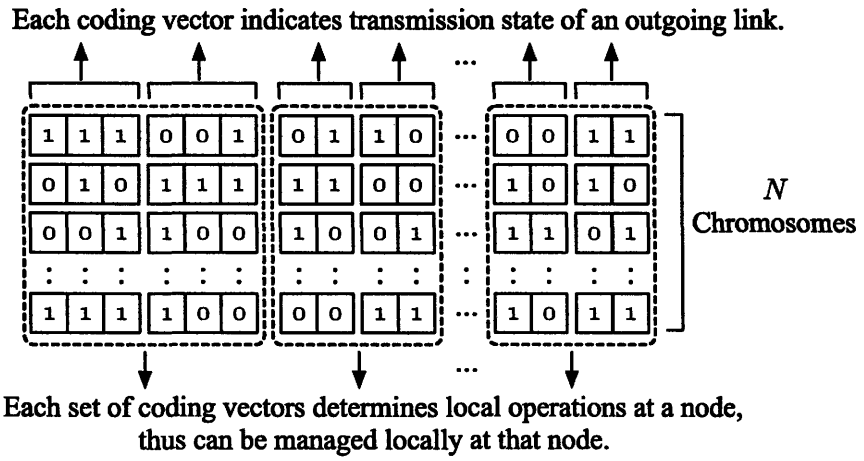


Figure 4-1: Structure of population.

Another key idea enabling the algorithm's efficient operation over the network is that a large number of chromosomes can be handled together by a single packet transmission. In most network coding solutions (e.g., [8]), a *global encoding vector* consisting of the coefficients that indicate the overall effect of network coding relative

to the source data is typically assumed to be carried in the packet header, whereas the payload conveys the actual data encoded using the coefficients. Here, we only need to transmit those coefficients for fitness evaluation without the data to be encoded, hence we can fill the payload with as many such coefficients as can be accommodated within a single packet.

As will be discussed below, the temporal distribution enables the most time consuming task, fitness evaluation, to be distributed over the network such that the computational complexity required at each node depends only on local parameters.

The most important benefit of the spatial distribution is that the coding resource optimization can be done on the fly while a network is operational, allowing for the following network coding protocol: As the source node sends an "optimize" signal with some initialization parameters, all the nodes participating in the multicast go into the optimization mode, running the algorithm described below in Section 4.1.2. As the distributed evolutionary algorithm proceeds, the links/nodes where coding is not required are identified. At the end of the algorithm, the source node sends a "transmit" signal that contains the index of the best chromosome in the last population and then the network starts to multicast data according to the best chromosome, in which coding is done only at the required links/nodes.

4.1.1 Assumptions

We assume that the end-nodes of each link can send some amount of feedback data in the reverse direction to the start-node of the link. We also assume that each interior node operates in a burst-oriented fashion; i.e., for the forward (backward) evaluation phase, each node starts updating its output only after an updated input has been received from all incoming (outgoing) links, similarly as in the generational approach in [8].

4.1.2 Implementation Details

The overall flow of our spatially distributed algorithm is shown in Fig. 4-2 with the location of each procedure specified. The most notable difference is that now the fitness evaluation is done in three steps: 1) forward evaluation phase from source to each receiver 2) backward evaluation phase from receivers to source and 3) final fitness calculation at the source.

```
[D1]  initialize population; (merging nodes)
[D2]  run forward evaluation phase; (all nodes)
[D3]  run backward evaluation phase; (all nodes)
[D4]  calculate fitness; (source)
[D5]  while termination criterion not reached (source)
      {
[D6]    calculate coordination vector; (source)
[D7]    run forward evaluation phase; (all nodes)
[D8]    perform selection, crossover, mutation; (merging nodes)
[D9]    run backward evaluation phase; (all nodes)
[D10]   calculate fitness; (source)
      }
[D11] perform greedy sweep; (all nodes)
```

Figure 4-2: Flow of spatially distributed algorithm.

The source node initiates the algorithm by transmitting a packet containing the following predetermined parameters: target multicast rate R, population size N, the size q of the finite field to be used, crossover probability, and mutation rate. Each participating node that has received the signal passes the signal to its downstream nodes.

Population Initialization [D1]

A merging node with $d_{in}(\geq 2)$ incoming and d_{out} outgoing links has to manage d_{out} coding vectors of length d_{in} for a single chromosome. Hence, for the population of size N, the node must have $N \cdot d_{out}$ coding vectors to determine the operations at that node. For initialization, each merging node generates $N \cdot d_{out}$ coding vectors randomly and independently, while setting the first one to an all-one vector.

Forward Evaluation Phase [D2, D7]

For the feasibility test of a chromosome, each node communicates a vector consisting of R components, which we refer to as a *pilot vector*. Each component of the vector is an element from the designated finite field $\mathbb{F}_q$ and the i-th ($1 \leq i \leq R$) component represents the coefficient used to encode the i-th source message. We assume that a set of N pilot vectors is transmitted together by a single packet. The source node initiates forward evaluation phase by sending out on each of its outgoing links a set of N random pilot vectors. Non-merging nodes simply forward the pilot vectors received to their outgoing links.

Each merging node transmits on each of its outgoing links a random linear combination of the received pilot vectors, computed based on the node's coding vectors as follows. For a particular outgoing link, let us denote the associated d_{in} coding vectors by v_1, v_2, ..., $v_{d_{in}}$. For the i-th ($1 \leq i \leq N$) output pilot vector u_i, we denote the i-th input pilot vectors received form the incoming links by w_1, w_2, ..., $w_{d_{in}}$. Define the set J_i of indices as

$$J_i = \{1 \leq j \leq d_{in} | \text{ the } i\text{-th component of } v_j \text{ is } 1\}. \tag{4.1}$$

Then,

$$u_i = \sum_{j \in J_i} w_j \cdot \text{rand}(\mathbb{F}_q), \tag{4.2}$$

where $\text{rand}(\mathbb{F}_q)$ denotes a nonzero random element from $\mathbb{F}_q$. If the set J_i is empty, u_i is assumed to be zero.

Backward Evaluation Phase [D3, D9]

To calculate the chromosome's fitness value, two kinds of information need to be gathered: 1) whether each receiver node can decode data of rate R and 2) how many nodes perform coding operations. After receiving all the pilot vectors, each receiver node can determine whether R messages are decodable for each of the N chromosomes

by computing the rank of the collection of received pilot vectors.

For the feedback of this information, each node transmits a vector consisting of N components, which we refer to as a *fitness vector*. Each of the components must be at least $\lceil \log(|V|+2) \rceil$ bits long since for each chromosome the number of coding nodes can range from zero to $|V|$ and an additional symbol (infinity) is needed to signify infeasibility. The backward evaluation phase proceeds as follows:

- After the feasibility test of the N chromosomes, each receiver node generates a fitness vector whose i-th ($1 \leq i \leq N$) component is zero if the i-th chromosome is feasible at the receiver node, and infinity otherwise. Each receiver node then initiates the backward evaluation phase by transmitting its fitness vector to all of its parents.

- Each interior node calculates its own fitness vector whose i-th ($1 \leq i \leq N$) component is 1 if it has any coding link for the i-th chromosome and 0 otherwise, then adds to it the sum of all the i-th components of the received fitness vectors. Each node then transmits the calculated fitness vector to *only one* of its active parents which we define as the nodes that have transmitted nonzero pilot vectors during the forward evaluation phase.

Note that, since the network is assumed to be acyclic, each coding node, for each chromosome, contributes exactly once to the corresponding component of the final fitness vector calculated at the source node. Hence, the above update procedure provides the source with the correct total number of coding nodes.

Fitness Calculation [D4, D10]

The source node calculates the fitness values of N chromosomes simply by performing component-wise summation of the received fitness vectors. Note that if an infinity were generated by *any* of the receivers, it should dominate the summations all the way up to the source, and thus the source can calculate the correct fitness value for infeasible chromosomes.

Termination Criterion [D5]

The source node can determine when to terminate the algorithm by counting the number of generations iterated thus far. As a practical concern, we may use an additional termination criterion such that the iteration is terminated if no progress is made in the best fitness value of the population for another predefined number of generations.

Coordination Vector Calculation [D6]

Since the population is divided into subsets that are managed at the merging nodes, genetic operations also need to be done locally at the merging nodes. However, some amount of coordination is required for consistent genetic operations throughout all the merging nodes; more specifically, for 1) consistent selection of chromosomes, 2) consistent paring of chromosomes for crossover, and 3) consistent decision on whether each pair is subject to crossover. This information is carried by a *coordination vector* consisting of the indices of selected chromosomes that are randomly paired and 1-bit data for each pair indicating whether the pair needs to be crossed over. Note that all this can be calculated at the source node based on the fitness values. The coordination vector is *transmitted together with* the pilot vectors in the next forward evaluation phase, hence there is no need for additional broadcast session for sharing the information.

Genetic Operations [D8]

Based on the received coordination vector, each merging node can locally perform genetic operations and renew its portion of the population as follows:

- For selection, each node only retains the coding vectors that correspond to the indices of selected chromosomes.
- For bit-wise/vector-wise crossover, each node independently determines whether each bit/coding vector is crossed over. Since no coding vector is shared by multiple merging nodes, this can be done independently at each merging node.

- For bit-wise/vector-wise mutation, each node independently determines whether each bit/coding vector is mutated without any coordination with other nodes.

Greedy Sweep [D11]

In contrast to other procedures described so far, greedy sweep requires more extensive coordination among the network nodes. Give that the benefit of greedy sweep may become negligible in the case of the vector-wise chromosome representation and genetic operators, one may well leave this procedure out. Nevertheless, we describe how the greedy sweep operator proceeds in our distributed framework, omitting the details of some data structures, if it is only required for this operator.

- After the iteration is over, the source node sends out to all nodes the index of the best chromosome in the last population.

- Upon receiving the index of the best chromosome, each merging node inspects its coding vectors corresponding to the received index. If the node has at least one coding link, it sends to the source node a packet containing the total length of the coding vectors associated with its coding link(s).

- After receiving the packets from the merging nodes that involve at least one coding link, the source node initiates the evaluation phase (forward+backward), as many times as the number of such received packets. Each time, the source node designates one merging node for possible flipping of the remaining 1's to 0 in its coding vectors.

- When initiating each evaluation phase, the source node includes the identifier of the currently considered merging node, which we refer to as node v, as well as the number of 1's in node v's coding vectors that indicate coded transmission, which we denote by m.

- Once the evaluation phase is initiated for node v, it is repeated m times. For the i-th (i=1, ..., m) time, node v performs the fitness evaluation as if the population consists of a single chromosome corresponding the the best index,

but the i-th remaining 1 is replaced with 0. All other merging nodes than node v perform the fitness evaluation as if the population consists of a single chromosome corresponding to the best index without any change.

- No selection or genetic operations are performed at any node.

- After the backward evaluation phase proceeds the same as before, each time the source node can check whether flipping the i-th (i=1, ..., m) variable would make the chromosome infeasible. This result is added to the pilot vector each time, according to which node v decides whether to keep the i-th (i=1, ..., m) remaining 1 flipped or not.

4.1.3 Complexity

Let us consider the computational complexity required for the fitness evaluation of a single chromosome. If the algebraic fitness evaluation is performed in a centralized fashion by calculating the transfer matrices to verify their nonsingularity, the required complexity is $O(|T| \cdot (|E|^{2.376} + R^3))$.

In the spatially distributed approach, for the fitness evaluation of a single chromosome, each merging node v computes random linear combinations of inputs in the forward evaluation phase, which requires $O(d_{in}^v \cdot d_{out}^v \cdot R)$, and each non-merging node w simply forwards the received data, which requires $O(d_{out}^w)$. Feasibility test at each receiver node t is done by calculating the rank of a $d_{in}^t \times R$ matrix, where we assume $d_{in}^t \geq R$, hence it requires $O({d_{in}^t}^2 R)$. In the backward evaluation phase, update of a fitness vector takes $O(d_{in}^v + d_{out}^v)$. Therefore, the computational complexity required for evaluation of a single chromosome is $O(\sum_{v \in \mathcal{V}} d_{in}^v d_{out}^v R + \sum_{w \in V \setminus \mathcal{V}} d_{out}^w + \sum_{t \in T} {d_{in}^t}^2 R)$. Note that this is substantially less than the computational complexity required for the centralized implementation of the algorithm.

4.2 Temporal Distribution

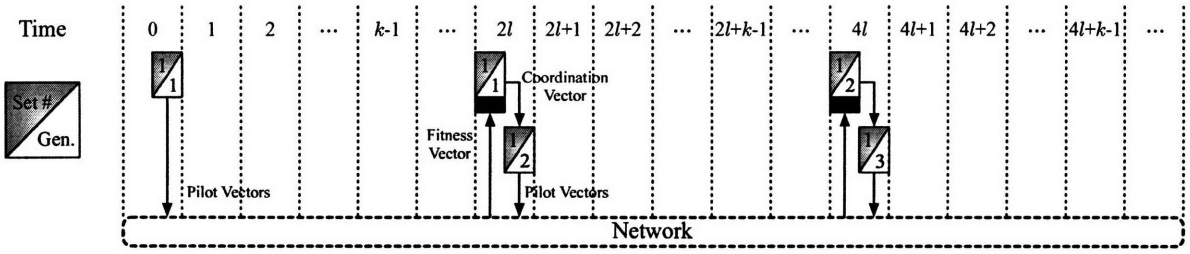
For the spatially distributed version of our algorithm presented in the previous section, which we refer to as **"Algorithm S"** hereafter in this section for notational convenience, once each generation is initiated at the source (procedure [D6] in Figure 4-2), the fitness values of N chromosomes become only available after the forward and backward evaluation phases are done, i.e., when the last fitness vector arrives at the source. The motivation for the second, *temporal*, distribution is to make more efficient use of the computational resources in the network by minimizing their idle duration during the GA iteration.

Let us assume that the time required for each node to calculate its outgoing pilot vectors based on the received ones is negligible compared with the time required for packet transmissions. Then, if we denote by l the length of the longest path from the source to any of the receiver nodes, the time lag between the initiation of the generation and the termination of the backward evaluation phase is $2l$ time units.

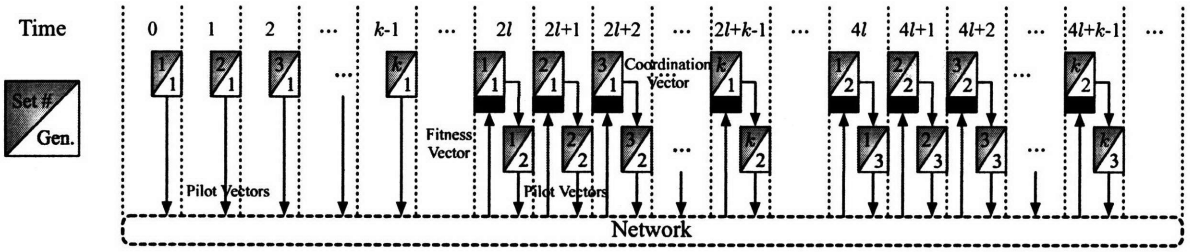
If we define the *evaluation efficiency*, which we denote by ε_v, as the number of fitness evaluations performed per unit time throughout the iteration of the GA, ε_v of Algorithm S is only $N/2l$ (see Figure 4-3(a)).

For better efficiency, we may still utilize the network resources, while waiting for the fitness vectors to return to the source, to evaluate more chromosomes. Suppose that, after initiating the forward evaluation phase of the n-th generation at time t, we initiate additional $k-1$ forward evaluation phases at times $t+1$, ..., $t+k-1$. When $k = 2l$, the network resources become fully utilized by the time when the fitness values of the first set of N chromosomes are available. Note that in fact k may even exceed $2l$, but then the evaluation of the $(n+1)$-th generation starts delayed at time $t+k$, rather than $t+2l$. For simplicity, we assume $k \leq 2l$ in the following.

As we introduce these additional evaluation phases, we may take three different approaches as follows. Note that any of the three approaches can be implemented together with the spatially distributed structure introduced in the previous section, with slight changes in the coordination vector.



(a) Timing diagram of Algorithm S (only spatially distributed)



(b) Timing diagram of Algorithm G/M (both spatially and temporally distributed; Generational/ Multiple populations).

Figure 4-3: Comparison of Algorithms S and G/M via timing diagrams.

4.2.1 Generational / Single Population

If we consider the k sets of N chromosomes as a single population, we have to wait additional $k-1$ time units, after the first backward evaluation phase ends (at time $t+2l$), to proceed to the next generation. In other words, we must flush the pipeline (and prime it again). Hence, the evaluation efficiency is given by

$$\varepsilon_v = \frac{kN}{2l+k-1},$$

whose maximum is obtained when $k = 2l$ such that $\varepsilon_v = \frac{2lN}{4l-1} \approx \frac{N}{2}$. For later comparison, we refer to this algorithm with $k = 2l$ as **"Algorithm G/S"**.

Avoiding the inefficiency of flushing the pipeline would generate a better ε_v and consequently faster convergence, provided that the algorithm requires a similar number of evaluations for the solutions of the same quality. Depending on how to manage those k sets of N chromosomes, we may consider two different approaches as follows.

4.2.2 Generational / Multi-Population

In this approach, referred to as **"Algorithm G/M"**, we regard each of those k sets of N chromosomes as a *subpopulation* which occasionally exchanges individuals with other subpopulations. It is worth to point out that, unlike typical island parallel GAs [5] where subpopulations are *spatially* distributed over different locations of computation, we have subpopulations that are *temporally* distributed over different times of evaluation.

We assume that migration is done at every f generations such that, before selection, each subpopulation replaces its worst $k-1$ individuals with the collection of $k-1$ individuals, one from each of the other $k-1$ subpopulations. Since we have no constraint on the (spatial) connections between the subpopulations, we can freely choose to assume and exploit the complete connectivity between subpopulations.

On the other hand, our algorithm imposes a different kind of constraint on migration, which is regarding the time synchronization between subpopulations. Let us assume that there is no delay in the network, so the backward evaluation phase of a particular subpopulation ends exactly after $2l$ time units its forward evaluation phase started. Suppose now that migration is about to happen at time $t+1$ while constructing the first subpopulation for the $(n+1)$-th generation. At that time, only the first subpopulation has the fitness values for the n-th generation, while all other $k-1$ subpopulations still wait for their fitness values for the n-th generation to become available. Similarly, at time $t+j$ $(1 \leq j \leq k)$, only the first j subpopulations have their fitness values for the n-th generation, while the remaining $k-j$ subpopulations do not. If we choose to perform migration in a *age-synchronized*, i.e., *temporally consistent* manner such that all the subpopulations exchange the best individuals of the *same* generation, we have to wait until time $t+k$ without being able to renew any subpopulation. Hence, we alternatively perform the *age-mixed*, i.e., *temporally closely consistent*, migration, where we collect the best individuals from the other $k-1$ subpopulations of the *most recent* generation for which the fitness values are available. For instance, when we renew the j-th $(2 \leq j \leq k-1)$ subpopulation at

time $t+j$, we take the best individual from each of the $1, ..., (j-1)$-th subpopulations at generation n, and from each of the $(j+1), ..., k$-th subpopulations at generation $n-1$.

Algorithm G/M proceeds in a completely pipelined manner (see Figure 4-3(b)), yielding the evaluation efficiency

$$\varepsilon_v = \frac{gkN}{(g+1)2l+k-1},$$

where g is the number of generations at the termination of the iteration. Note that, when $k = 2l$ and $g \gg 1$, $\varepsilon_v \approx N$.

4.2.3 Non-Generational

Rather than managing k separate subpopulations, this approach, referred to as **"Algorithm NG"**, operates on a single population of size $M = kN$. The population is updated when the fitness values of each of the k sets of N chromosomes, referred to as *offspring*, become available (i.e., "just-in-time"). This is a temporally "sloppy" approach. From time 1 to k, the forward evaluation phases for the initial (random) k offspring are initiated. At time $2l+j$ $(1 \leq j \leq k)$, the fitness values for the j-th offspring can be calculated at the source and *all* those N chromosomes are just added to the population. We then calculate the coordination vector for the j-th offspring, by performing tournament selection out of the current population, which is partially filled until time $2l+k$, and initiate the forward evaluation phase for the second generation. At time $4l+j$ $(1 \leq j \leq k)$ and on, we update the population as follows: First combine the j-th offspring, whose fitness values are just calculated, with the existing population, and then pick the best kN individuals, out of those $(k+1)N$ individuals, to form the updated population.

Considering each window of $2l$ time units from the beginning, we notice that except for the first and the last windows, kN chromosomes are evaluated in each window (see Figure 4-4). Hence, if we assume that the total number of elapsed time units is large ($\gg 1$), we have $\varepsilon_v \approx \frac{kN}{2l}$, and when $k = 2l$, we obtain the maximum

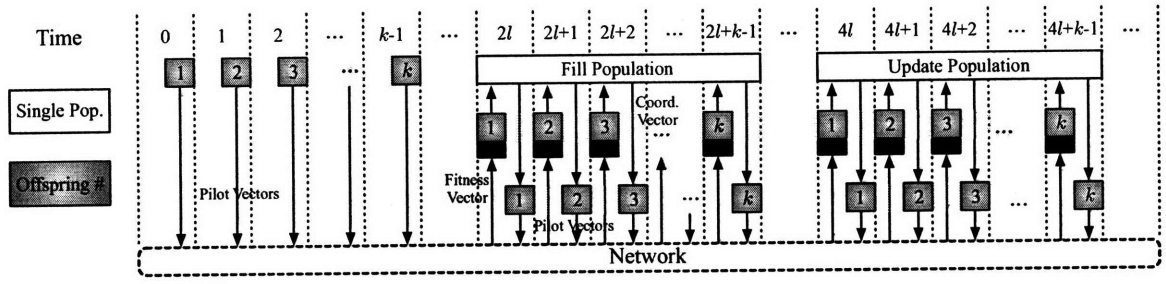


Figure 4-4: Timing diagram of Algorithm NG (both spatially and temporally distributed; Non-Generational).

$\varepsilon_v \approx N$.

4.3 Population Sizing with Distributed Implementation

As pointed out in Section 3.6, it is not an easy task to predict the accurate population size required for a specific problem. However, it is always desirable to allow some level of flexibility in adopting a large-sized population when needed, without incurring too much complexity overhead.

In our distributed framework, the population size can be adjusted mainly by modifying the size of the packets to be used for fitness evaluation. The key observation is that the length of a pilot vector depends on R, the desired multicast rate represented as a multiplicative factor *relative* to the unit capacity, but not the actual data rate which amounts to R times the data rate corresponding to a unit capacity. Hence, a large number (typically several hundreds) of pilot vectors can possibly be handled with a single packet transmission.

Example 3 *Consider network B-7 introduced in Section 3.7.1, which is constructed by cascading a number of copies of network B (Fig. 1-1(b)). To determine the size of a single pilot vector for this network, we first choose the size of the finite field upon which we construct the random linear code. The limiting factor on the field size is the error probability of the randomized feasibility test, whose upper bound is given by*

$1-(1-d/q)^{\nu}$, *where* ν *is the maximum number of links in any set of links constituting a flow solution from the source to any receiver [27]. In* G, $d = 8$ *(number of receiver nodes) and* $\nu = 12$. *If we desire to keep the error probability below 0.01, the smallest power* q *of 2 that meets the error bound is 14 and thus the length of* N *pilot vectors is* $NR\log_2 q = 28N$ *bits. Also, the length of the fitness vector is* $N \cdot \lceil \log_2(|E|+2) \rceil = 7N$ *(*$|E| = 70$*) bits, and the length of the coordination vector is* $N(\lceil \log_2 2N \rceil)$ *bits.*

For example, if the unit packet size is 1500 bytes (the maximum Ethernet packet size), the largest N *such that* N *pilot vectors and a coordination vector fit into a single packet turns out to be 321.* □

As the population size N varies, the size of memory used at each node to store the chromosomes must also be adjusted accordingly. Also, the computational complexity required at each node during the forward and backward evaluation procedures, as well as the genetic operations, scale linearly with N. However, since each node stores only the relevant portion of the chromosomes and also the computation at each node involves only that portion of the chromosomes, the impact of increased N may be considered insignificant relative to its impact on the packet transmission.

4.4 Performance Evaluation

4.4.1 Effect of Spatial Distribution

As described in Section 4.1.3, the computational complexity required by the spatially distributed algorithm (Algorithm S) depends only on local topological parameters, which can often lead to a significant gain in terms of the running time compared with the our algorithm implemented in a centralized fashion (which we refer to **"Algorithm C"**). To compare the elapsed running time of the two versions of the algorithm, we run a test on a created set of topologies with high connectivity such that there exists a link between each pair of numbered nodes i and j $(i < j)$, where the source is node 1 and the receiver nodes are the last 10 nodes.

The test is done by a simulation on a single machine while each node's function is

performed by a separate thread, thus it is pessimistic since it cannot benefit from the multi-processing gain whereas it only suffers from additional computational burdens for managing a number of threads. Table 4.1 shows that, nevertheless, Algorithm S exhibits an advantage in running time over Algorithm C as the size of the network grows.

Table 4.1: Running time per generation (seconds)

Number of nodes	15	20	25	30	35	40
Algorithm C	0.3	1.5	4.3	13.5	29.5	65.6
Algorithm S	1.8	2.7	4.4	6.3	10.8	15.4

4.4.2 Effect of Temporal Distribution

To compare the different versions of the temporally distributed structure of our algorithm, we use network B-15 introduced in Section 3.7.1. We assume that N, the number of chromosomes handled by a single packet, is around 200, which is a reasonable value when the unit packet size is set to 1500 bytes as a typical ethernet packet. Since $l = 16$ in network B-15, $k = 2l = 32$.

Table 4.2: Population parameters for algorithms.

	Parameters on Population
Algorithm S	Pop. size: 200
Algorithm G/S	Pop. size: 6400
Algorithm G/M_{10}	Subpops. (size, #): (200,32), Migration freq.: 10
Algorithm G/M_1	Subpops. (size, #): (200,32), Migration freq.: 1
Algorithm NG	Pop. size: 6400, Offspring size: 200

Table 4.2 summarizes the parameters for five algorithms we experiment with. Migration frequency (f) is changed from 10 to 1 from Algorithm G/M_{10} to G/M_1. We set the tournament size to the half of the (sub)population size in each algorithm, i.e., 100, 3200, 100, 3200 for Algorithms S, G/S, G/M, NG, respectively. We perform 30 runs for each algorithm until the algorithm finds the optimal solution, which for network B-15 is known to be zero.

Table 4.3 shows the average number elapsed time units averaged over 30 runs. For better comparison, we also display in the table the *time efficiency* ε_t defined as the algorithm's speedup with respect to Algorithm S. Also shown in the table are the total number of fitness evaluations, i.e., the number of evaluated chromosomes until the algorithm terminates, and the evaluation efficiency ε_v obtained from the experiments, which indeed matches the theoretical values almost exactly. For elapsed time and number of evaluations, p-value resulting from paired t-test with the next best (i.e., smallest) one is reported.

Table 4.3: Result of experiments.

	Time	p-value	ε_t	#Eval	p-value	ε_v
S	13,907	-	1.00	86,920	1.38e-14	6.25
G/S	5,427	1.66e-08	2.56	542,720	2.10e-03	100.00
G/M_{10}	2,497	1.58e-04	5.57	492,920	0.307	197.44
G/M_1	4,157	7.55e-03	3.35	824,980	-	198.46
NG	3,968	0.691	3.50	781,100	0.691	198.39

Pipelining is intended to be efficient by reducing the idle time of network nodes, hence Algorithm S, which does not pipeline, has the lowest ε_v. Though Algorithm G/S, which pipelines but stop to flush and re-prime, has much increased ε_v, Algorithms G/M_{10}, G/M_1, and NG, which operate fully pipelined, offer the highest ε_v. Note, however, that the different dynamics of these algorithms may impact the number of fitness evaluations required to reach the optimal solution, hence as can be observed in Table 4.3, the number of evaluations (and consequently, the realized ε_t) do not reveal ε_v in proportion. Figure 4-5 shows that evaluation efficiency comes at the cost of additional fitness evaluations. Algorithms G/M_{10} and S dominate all others yet not each other; Algorithm S is less efficient (it does not pipeline) but requires less fitness evaluations, while G/M_{10} is more efficient but requires more evaluations. Algorithm G/M_{10} gives a speedup (ε_t) of more than 5 times over Algorithm S. Algorithms G/S, G/M_1 and NG, though dominated by G/M_{10}, still offer higher ε_t than Algorithm S.

Algorithms G/M_1 and G/M_{10}, though distributed temporally, resemble a spatially distributed GA (referred to as multiple-deme GA in [5]) in that they incur

no communication overhead and can assume a fully-connected processor topology. The only difference in algorithm dynamics is that migration takes place between sub-populations that differ in age by one generation (see Section 4.2.2). Thus the performance of Algorithms G/M_1 and G/M_{10}, as compared with that of Algorithm S, is in fact foreseeable from the observation that, in general, multiple-deme GAs require a greater number of evaluations than a standard GA while offering speedups due to parallelism, which is equivalent to higher ε_v. However, in our experiments, the size and the number of subpopulations are determined to maximize ε_v rather than the performance of GA. Determining the migration strategy for multiple-GAs is an open question and probably problem dependent [4].

Algorithm NG is a completely new algorithm, where the selection from the population and the replacement of offsprings are temporally inconsistent. A (slightly) similar property can be found in the second prototype for parallel GA in [20], where the algorithm sends out individuals to processors to be evaluated, and inserts and re-selects them opportunistically, i.e., when their fitness becomes available. Such, rather radical, changes in algorithm dynamics may raise a question whether Algorithm NG would even work, which is verified by our experiments. The performance of Algorithm NG is similar to that of G/M_1, hence surpassed by G/M_{10}, which can be explained by the observation that the temporal mixing of Algorithm NG is similar to Algorithm G/M_1's frequent mixing. Together, these two results suggest that the temporally distributed algorithm is robust to age mixing (i.e., temporal sloppiness), which may deserve further in-depth analysis in the future.

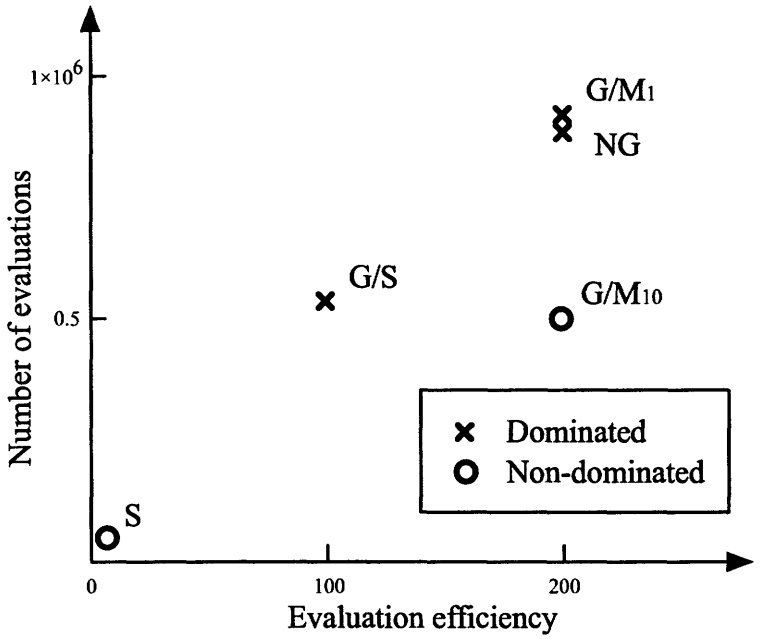


Figure 4-5: Evaluation efficiency vs. number of evaluations.

Chapter 5

Multiple Objectives

In this chapter, we investigate the issue of the tradeoff between network coding and link usage in multicast network coding. As introduced in Chapter 1, network coding makes minimum-cost multicast, an NP-complete problem with traditional routing alone, polynomially solvable, but if we consider the network coding capability as another resources to optimize, the link cost is actually minimized at the expense of the coding cost.

However, as will be discussed later, determining whether there exists such a tradeoff turns out to be NP-hard. We thus extend our evolutionary algorithm to investigate the tradeoff. One way to address the problem indirectly is to combine the coding and links costs into a single objective and then to minimize it using our evolutionary algorithm as presented in the previous chapters. Such a method, however, can only provide a single solution that minimizes the combined cost. Alternatively, we may tackle the problem by repeatedly applying a two-stage method where in the first stage we fix the set of the links to be used and then determine a minimal set of coding nodes. This type of method, however, can be inefficient since the number of possible combinations of the links can be very large.

Evolutionary algorithms, in fact, can serve much more effectively as a method to identify such tradeoffs [9]. In this chapter, we generalize our approach to investigate the utility of network coding in comparison with the amount of saved link cost, which has been unable to measure so far with any other method.

5.1 Problem Formulation and Related Work

Let us proceed to describe the problem based on the same network model and assumptions given in Section 2.1. In addition, each link $e \in E$ is assigned link cost l_e, which is incurred when the link is used for transmission, and coding cost c_e, which is incurred if the transmission on the link involves network coding rather than simple forwarding.

Let $f_c(x)$ and $f_l(x)$ denote the total coding and link costs, respectively, for any scalar linear network code x. We then wish to find the *Pareto optimal front* (which can be shown to be unique) defined as follows:

Definition 1 *The Pareto optimal front is the set of the cost pairs* (f_c^*, f_l^*) *of a feasible transmission scheme such that there exists no other scheme* x *that is feasible and satisfies* $\{f_c(x) < f_c^*, f_l(x) \leq f_l^*\}$ *or* $\{f_c(x) \leq f_c^*, f_l(x) < f_l^*\}$.

Theorem 3 *Finding the Pareto optimal front between the coding and link costs for multicast is NP-hard.*

Proof: The problem to determine whether employing network coding at more nodes/links decreases link cost is NP-hard, to which the NP-complete problem of computing the minimum cost for multicast without network coding [50] is reduced. To decide the converse, i.e., whether removing links increases the minimum number of coding nodes/links, is also NP-hard (Theorem 1). □

For optimization with multiple objectives, i.e., the coding and link costs in our case, a number of algorithms have been proposed to obtain the Pareto optimal front in a single run [9]. Commonly in those algorithms, if solution x is inferior to another solution y with respect to one or more of the optimization criteria while the two are the same in all the remaining criteria, x is said to be *dominated* by y. More formally, in the case of the minimization problem with the two objectives f_c and f_l, the notion of domination is defined as follows:

Definition 2 *For chromosomes* x *and* y, x *is dominated by* y *(or* y *dominates* x*) if either* $\{f_c(x) > f_c(y), f_l(x) \geq f_l(y)\}$ *or* $\{f_c(x) \geq f_c(y), f_l(x) > f_l(y)\}$ *holds.*

Multi-objective GAs share largely the same structure with ordinary simple GAs, with some notable differences in the selection mechanism. Multi-objective selection mechanisms employ various algorithmic techniques to locate the resulting population as close to the actual Pareto optimal front as possible [9]. First, while selection in simple GAs puts more weights on the solutions with better fitness values with respect to a single objective, multi-objective GAs employ a mechanism that assigns higher probabilities for selection to less dominated solutions. In addition, multi-objective GAs employ the mechanism that preserves diversity among the solutions in an effort to obtain the full Pareto optimal front. Out of many existing multi-objective GAs, we primarily focus on Deb et al.'s NSGA-II [9], based on which we implement a novel selection mechanism specific to our problem.

5.1.1 Selection Mechanism of NSGA-II

In the original NSGA-II [9], selection is done based on two criteria: non-domination rank and crowding distance. After calculating the fitness values, the algorithm finds the first *non-domination front* $\mathcal{F}_1$, the set of the chromosomes that are *not* dominated by any other chromosome, assigning *non-domination rank* 1 to those chromosomes. For $i \geq 2$, i-th non-domination front $\mathcal{F}_i$ consists of the chromosomes in $P \setminus \{\mathcal{F}_1 \cup \cdots \cup \mathcal{F}_{i-1}\}$ that are not dominated by others, to which non-domination rank i is assigned. It can be shown that each $\mathcal{F}_i (i \geq 1)$ is nonempty unless we exhaust all the chromosomes, and thus there are only a finite number of non-domination fronts.

The chromosomes belonging to the same non-domination front are sorted with respect to each of the objectives, one after another, and the crowding distance of each chromosome, which is defined as the sum of the differences between its next better and next worse chromosomes along each axis of the objectives, is calculated. Intuitively, crowding distance is a measure of the densities on each non-domination front such that a chromosome in the sparse region has a high distance.

Let P_t denote the population at generation t. Then, an intermediate population Q_t of the same size N is created using a binary tournament selection, where we repeat the following procedure until Q_t is filled: a random pair of chromosomes is chosen and

the one with a lower non-domination rank is selected, or if the ranks are the same, the one with a higher crowding distance is selected. After calculating the fitness values of Q_t, the N best chromosomes out of $P_t \cup Q_t$ are selected for the actual population P_{t+1} for the next generation $t+1$. At the end of the algorithm, $\mathcal{F}_1$ of the last population is the resulting Pareto optimal front.

It is shown in [9] that non-domination rank and crowding distance can be efficiently calculated in $O(MN^2)$ time, where M is the number of objectives and N is the population size. This NSGA-II is then evaluated to show that it surpasses other multi-objective optimization algorithms in terms of minimizing various classes of continuous-valued functions.

5.2 Problem Specific Selection Mechanism for Multiple Objectives

As motivated in Section 2.4 and evidenced subsequently in Section 3.7.3, utilizing even a very simple problem-specific knowledge, such as the modularity among variables imposed by the given network topology, can lead to a substantial gain in the algorithm's performance.

In this section, we discuss the unique characteristics of our problem that can be incorporated into the framework of multi-objective GAs to obtain a more close-to-optimal Pareto front.

5.2.1 Different Convergence Time

Our problem displays some unique characteristics that cannot be well handled with the above selection mechanism alone. One of the hurdles in applying the above selection mechanism to our problem is that it is likely to produce only a part of the actual Pareto optimal front. More specifically, the resulting final front has a strong tendency to be skewed toward the low link cost region of the desired front.

Example 4 *Let us consider network B-7 introduced in Section 3.7.1, which is con-*

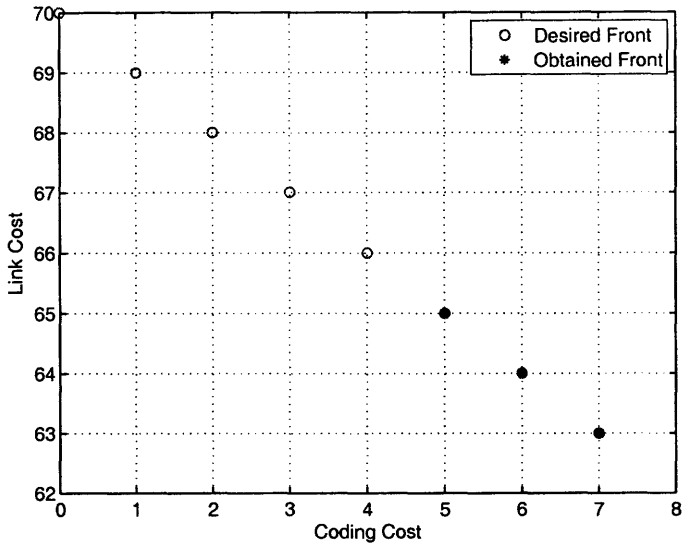


Figure 5-1: Skewed Pareto front for Network B-7 calculated by NSGA-II.

structed by cascading a number of copies of network B (Fig. 1-1(b)). Assume that the coding and link costs of each link are both assumed to be 1. For this network, the minimum number of coding links is known to be zero, but there exists a tradeoff opportunity for each copy of network B. Hence, the desired Pareto front is given as depicted in Fig. 5-1 by circles. If we apply the evolutionary algorithm as presented in Chapter 3 with the selection mechanism replaced by that in NSGA-II [9], we almost always obtain a skewed front as depicted in Fig. 5-1 by the stars in the lower right part of the desired full front. □

This phenomenon can be understood from the different difficulties of the optimization along the two objectives, i.e., while minimizing coding cost is NP-hard, minimizing link cost regardless of coding cost is polynomially solvable. In the context of GA, this difference translates into different convergence times to the optimal regions of the two objectives. Before the chromosomes with a low coding cost emerge, those having a low link cost are more likely to appear in earlier generations. For the same link cost, the chromosomes with a lower link cost dominate the ones with a higher link cost and thus are favored in the selection process. This selection pressure toward the low link cost region often dominates the early stage of evolution, making it hard for the population to evolve into the high link cost region which may eventually

lead to low coding costs.

Therefore, we need to ensure that the chromosomes having high link costs are not lost prematurely, because they may initially be dominated by the ones with lower link costs and the same coding cost, but eventually lead to the solutions with lower coding costs. To this end, we define the *coding front* $\mathcal{C}_1$ as the set of the chromosomes having the least coding cost for each link cost level. More specifically, for each chromosome $x \in \mathcal{C}_1$ there exists no feasible chromosome y that satisfies $\{f_c(y) < f_c(x),\ f_l(y) = f_l(x)\}$; $\mathcal{F}_1$ is in fact a subset of $\mathcal{C}_1$. We then assign non-domination rank 1 also to the chromosomes in $\mathcal{C}_1 \setminus \mathcal{F}_1$ so that those chromosomes are not lost prematurely.

5.2.2 Degeneracy

Evolutionary algorithms for multi-objective optimization, as mentioned above, are typically applied to optimizing various kinds of continuous-valued function for performance evaluation [9]. In most such cases, each chromosome is directly mapped to a different value of the functions' variable(s). In our problem, however, each chromosome determines the operations performed at the interior nodes and the two discrete-valued objectives are the resulting coding and link costs, which in a sense are expressed in a parameterized form of the chromosome. Hence, there can be a substantially large number of different chromosomes that correspond to the same coding and link costs, yielding many chromosomes with zero crowding distance.

Within the original selection mechanism, the chromosomes with zero crowding distance are those least favored on the same non-domination front. However, we observed from our experiments that two chromosomes with zero crowding distance may be very different, i.e., the Hamming distance between the two may be very large. In fact, having those chromosomes that are "neutral" with respect to fitness landscape but "diverse" in terms of Hamming distance on a non-domination front may promote finding even better recombined chromosomes.

Hence, for those chromosomes with zero crowding distance, we use Hamming distance as a secondary measure of distance. More specifically, suppose that chro-

mosomes x and y, having the same rank and zero crowding distance, compete for selection. We then compute $h(x)$, defined as the Hamming distance from x to the closest one among the remaining chromosomes having the identical objective values, and $h(y)$, defined similarly, and finally select the one with larger $h(\cdot)$.

5.3 Distributed Implementation

As mentioned in the previous section, the changes introduced as we generalize the algorithm to the multi-objective case are only concerned with the selection part. In the distributed framework described in Section 4.1, those changes translate into the modification of the data collected and distributed by the source node as well as some additional calculation at the source node, without disrupting the algorithm's main structure. Therefore, most part of the distributed implementation carries over to our generalized evolutionary approach for multi-objective optimization.

In this section, we describe the spatially distributed implementation of our multi-objective evolutionary algorithm, highlighting only the parts that require changes from the single-objective case. The overall flow of our multi-objective evolutionary algorithm is shown in Fig. 5-2, in which the greedy sweep operator is no longer performed as we consider both the coding and link costs. Note that the algorithm can also be distributed temporally by slightly changing the structure of the coordination vector, as suggested in Section 4.2.

Initialization [M1]

The initialization of the algorithm proceeds the same way as **[D1]** in Section 4.1.2 except that each node must store the intermediate population Q while performing genetic operations, thus the actual size of memory required at each node is $2Nd_{in}d_{out}$ bits.

Forward Evaluation Phase [M2, M7]

Forward evaluation phase is the same as **[D2, D7]** in Section 4.1.2.

```
[M1]  initialize; (all nodes)
[M2]  run forward evaluation phase; (all nodes)
[M3]  run backward evaluation phase; (all nodes)
[M4]  calculate fitness; (source)
[M5]  while termination criterion not reached (source)
      {
[M6]    calculate coordination vector; (source)
[M7]    run forward evaluation phase; (all nodes)
[M8]    perform selection, crossover, mutation; (interior nodes)
[M9]    run backward evaluation phase; (all nodes)
[M10]   calculate fitness; (source)
      }
```

Figure 5-2: Flow of distributed algorithm for multiple objectives.

Backward Evaluation Phase [M3, M9]

To calculate the fitness value of each chromosome, the source node now requires three, as opposed to two in the single objective case, kinds of information: 1) whether all the receivers can decode the data of rate R, 2) how many nodes perform coding operations, and 3) the total cost of the links used for (either coded or uncoded) transmission.

Each receiver can determine, by computing the rank of the collection of received pilot vectors, whether data of rate R is decodable for each of the N chromosomes. By inspecting its coding vectors used in the forward evaluation phase, each node can determine whether coding operations are done and calculate the link cost incurred at the node.

For the feedback of this information, each node transmits upstream a fitness vector consisting of N components, whose i-th component conveys the information needed to calculate the fitness value of the i-th chromosome. Each component of a fitness vector now contains two information: the coding cost and link cost up to the location where the fitness vector is generated. An infeasible chromosome is signified by the *infinite* coding and link costs. The remaining backward evaluation phase proceeds the same way as **[D3, D9]** in Section 4.1.2.

Fitness Calculation [M4, M10]

The source calculates the two fitness values, the coding and link costs, of N chromosomes simply by summing the received fitness vectors component-wise. Note that if an infinite cost were generated by *any* of the receivers, it should dominate the summations all the way up to the source, and thus the source can detect the infeasible chromosomes.

Termination Criterion [M5]

Termination criterion is the same as **[D5]** in Section 4.1.2.

Coordination Vector Calculation [M6]

Again, the information required for coordinated genetic operations at interior nodes is carried by a coordination vector calculated at the source. The coordination vector essentially conveys the outcome of the selection mechanism described in the previous section.

We now show how the selection mechanism is implemented in our distributed setup and how a coordination vector is constructed. Let us assume that, at generation t, the fitness values (not the actual chromosomes) of P_t and Q_t are available at the source, which will be shown to be valid later. From those fitness values indexed properly, the source node can calculate the non-domination rank and crowding distance of P_t and Q_t, and then determine the indices representing the chromosomes that comprise P_{t+1}. Note that, from just those indices, each interior node can retrieve its relevant portion of P_{t+1}, if the actual chromosomes of P_t and Q_t (without the fitness values) were stored at those interior nodes. Hence, the coordination vector consists of the indices of the selected chromosomes, permuted randomly, which thus provides the paring information for crossover as well. The coordination vector is then transmitted *piggyback* onto the pilot vectors during the next forward evaluation phase, without requiring an additional procedure dedicated to it.

Genetic Operations [M8]

Based on the received coordination vector, each node can locally perform genetic operations and renew its portion of the population. For selection, each node now retains, out of P_t and Q_t saved at the node, the coding vectors that correspond to the indices contained in the received coordination vector to construct P_{t+1}.

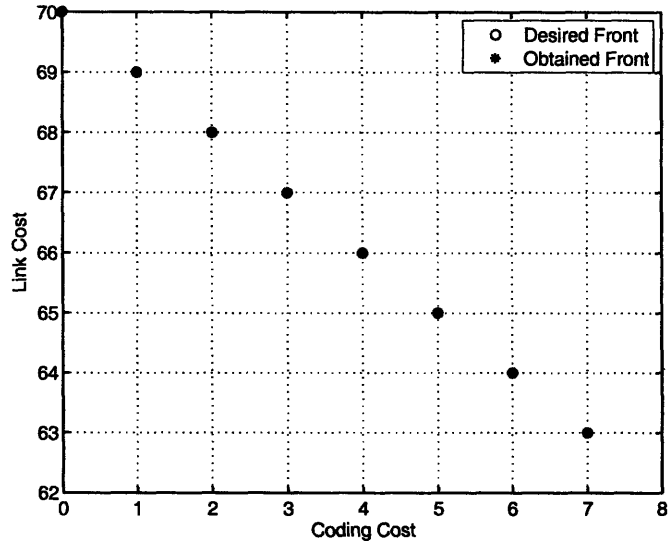
After crossover and mutation, which are performed in exactly the same way as [D7] in Section 4.1.2 (hence details are omitted), the relevant portion of Q_{t+1} is constructed at each node. The fitness values of Q_{t+1} start to be calculated as the algorithm proceeds to the forward evaluation phase of generation $t+1$. Note that since P_{t+1} was a subset of $P_t \cup Q_t$, the source node already had the fitness values of P_{t+1}, and at the time when the coordination vector for generation $t+1$ is constructed at the source node, the fitness values of Q_{t+1} will become available at the source, which validates the assumption we made in the coordination vector calculation procedure.

5.4 Performance Evaluation

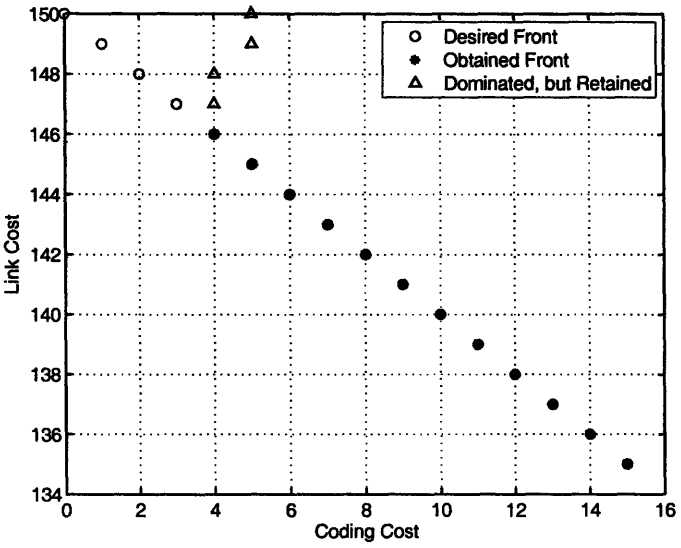
We demonstrate the performance of our algorithm by carrying out simulations on various network topologies, assuming each link has the unit coding and link costs. First, we apply our algorithm, combined with the selection mechanism described in Section 5.2, again to network B-7 as in Example 4 for which the Pareto optimal front is known. Note that, as depicted in Fig. 5-3(a), the algorithm now successfully finds the full Pareto optimal front. We also try a larger network B-15 of the same type, which is now a depth-4 binary tree containing 15 copies of network B (Fig. 1-1(b)) and has 16 receivers. The resulting is depicted in Fig. 5-3(b) marked by the stars while the circles represent the desired Pareto optimal front. Note that the points marked by triangles represent the chromosomes belonging to $\mathcal{C}_1 \setminus \mathcal{F}_1$, which, though dominated by others, are retained in the first front. Those chromosomes may eventually converge to the desired front if they were allowed more time for evolution.

We also evaluate the performance of our algorithm based on the two random topologies, R-50 and R-75, introduced in Section 3.7.1, having 50 and 75 nodes,

respectively. For network R-50, the obtained front consists of only a single point as depicted in Fig. 5-4. Note that this is in fact expected because the minimum coding cost without restricting the link cost was 2, as experimented in Section 3.7. Hence, from the resulting non-domination front, we find that network R-50 requires network coding at some links, but coding does not save the link cost. Network R-75 also produces a single-point non-domination front, implying that there is no tradeoff opportunity.



(a) Network $B-7$



(b) Network $B-15$

Figure 5-3: Calculated non-domination fronts using our problem specific selection mechanism.

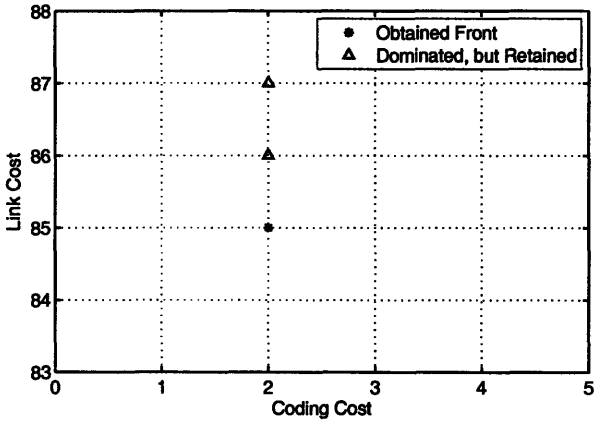


Figure 5-4: Non-Domination front for network $R-50$ showing no tradeoff.

Chapter 6

Network Applications

6.1 Integration of Network Coding into Heterogeneous Wireless Networks

In this section, we focus on wireless networks, in particular, the heterogeneous ad hoc wireless networks where a number of coding nodes are to be placed among legacy nodes that do not handle network coding operations well. Many interesting questions may arise regarding this scenario: How many coding nodes are required? Is network coding still required only at a small subset of nodes as we found in a generic network model? If only a small number of coding nodes are enough, where should those coding nodes be located? Can we fix their locations despite varying communication demands? How should those coding nodes interact with other non-coding nodes? Though providing complete answers for all such questions is a difficult task, we seek to better understand those questions based on our proposed evolutionary framework and provide some quantifiable results through various simulations.

Given the vulnerability of ad hoc wireless networks to various kinds of losses, we also discuss how our algorithm works in the presence of packet erasures which may be caused by delays, collisions, or topological changes. For this purpose, we utilize the temporally distributed structure of our algorithm, whose initial motivation was better utilization of computational resources over the network. We show that the temporally

distributed algorithm offers a significant advantage in overcoming the adverse effect of erasures.

6.1.1 Problem Formulation and Assumptions

We now generalize our network model to a directed acyclic hypergraph $H = (V, A)$, where V is the set of nodes and A is the set of hyperarcs. Each hyperarc (i, J) represents a broadcast link from node i to a non-empty subset of V. Hyperarcs have unit-capacity implying that a unit-size packet can be transmitted per unit-time in the given direction. We assume that the end-nodes of a hyperarc can send some amount of feedback data in the reverse direction to the start-node of the hyperarc. We again consider the single multicast scenario in which a single source $s \in V$ wishes to transmit data at a given rate R to a set $T \subset V$ of receiver nodes, where $|T| = d$. We assume that the given multicast rate R is achievable when network coding is allowed at all nodes.

We are interested in heterogeneous wireless networks where a number of coding nodes and legacy non-coding nodes coexist. The coding nodes that need to be placed among the legacy nodes are assumed to have some built-in mechanism for network coding operations, which makes the cost of coding negligible. On the other hand, we assume that the legacy nodes can provide only limited support for network coding at the application layer and that such emulated coding operations are computationally expensive.

Hence, our objective is now to minimize the number of the legacy nodes that have to emulate coding operations in order to achieve the desired multicast rate.

Even if there is no need for any node to know the whole topology, each node is assumed to know its neighboring nodes, i.e., upstream and downstream nodes. We assume that the mobility of the nodes is limited or at least slow that there is no change in the neighboring relations among nodes during the iteration of the algorithm.

6.1.2 Algorithm's Operation in Wireless Networks

Let us consider how our algorithm for coding resources optimization, spatially distributed as described in Section 4.1, operates in wireless networks, assuming first that the locations of the coding nodes are given. Later we will demonstrate how we can apply our proposed algorithm to investigate the issue of where to put the coding nodes and how they should interact with non-coding nodes.

First, we assume that the algorithm operates on top of appropriate mechanisms that handles the effect of lossy transmission or the issue of collision; the losses due to either erasures or collisions may in fact be resolved eventually through network coding. Then, our algorithm proceeds largely the same way as in Section 4.1, with the generic network links now replaced with the hyperarcs that account for wireless broadcast. The only change that needs to be made in the wireless case is the backward evaluation phase ([D3, D9] in Fig. 4-2) as follows:

- After the feasibility test of the N chromosomes, each receiver node generates a fitness vector whose i-th ($1 \leq i \leq N$) component is zero if the i-th chromosome is feasible at the receiver, and infinity otherwise. Each receiver then initiates the backward evaluation phase by *broadcasting* its fitness vector to all of its parents.

- Each interior node calculates its own fitness vector whose i-th ($1 \leq i \leq N$) component is 1 if it is a non-coding legacy node but performed coding operations for the i-th chromosome, and 0 otherwise, then adds to it the sum of all the i-th components of the received fitness vectors destined for that node. Each node then transmits the calculated fitness vector to *only one* of its active parents which we define as the nodes that have transmitted nonzero pilot vectors during the forward evaluation phase. All other parent nodes will still overhear the fitness vector but only use it as an acknowledgment rather than adding it to their own fitness vectors.

Now let us take into account the losses due to erasures, collisions, etc. Even if they may be resolved eventually through network coding, the algorithm's operation is

certainly affected when the packets carrying control signals are dropped. First of all, the flow of the algorithm may halt since each node expects the inputs from all of its neighboring nodes to proceed. We thus need to employ an expiration mechanism such that if some inputs are not received until the timer expires, each node just proceeds to the next stage without indefinitely waiting for those inputs.

There are further problems remaining even if we can prevent the halt of the algorithm at the expense of some possible delays. In particular, if packets are lost (and thus ignored) during the forward evaluation phase, feasible solutions may appear infeasible, hence the fitness values of the affected chromosomes may get mistakenly worse. On the other hand, packet losses in the backward evaluation phase may lead to incorrectly better fitness values because the signals for infeasibility may not be delivered to the source node or some coding nodes may be missed out during the counting process.

However, the built-in robustness of GA can significantly reduce the negative effect of such incorrect fitness values. First, GA operates on the set of candidate solutions (population), rather than a single one. Second, in search of the optimal solution, GA iteratively reconstructs and re-evaluates the population after mixing and perturbing high-performance solutions. Hence, even if the population may have corrupted fitness values at a particular generation, the algorithm may operate without much disruption because the affected chromosomes can be re-evaluated and corrected in the subsequent generations. It is worth to point out that the same argument also applies to the effect of slight topological changes.

Our algorithm may become even more robust by employing the temporally distributed structure introduced in Section 4.2. Though the main purpose of the temporal distribution was to promote more efficient use of network resources, we will show here that the temporally distributed structure leads to much more robust operation of the algorithm in the presence of losses.

Next, we investigate through simulations various issues regarding the questions we posed earlier.

6.1.3 Number of Coding Nodes

First, in order to find how many coding nodes are required, we ran our algorithm without distinguishing coding nodes and non-coding legacy nodes; i.e., there were no nodes where network coding is free. For the time being, losses are assumed to be negligible.

The simulations were performed based on random wireless networks generated as follows. Nodes are placed randomly within a 10×10 square with radius of connectivity 3. A unit-rate hyperlink is originated from each node toward the set of nodes that are within the connectivity range and have a higher horizontal coordinate than the node itself. The source node was chosen randomly among the nodes in the left half and the receiver nodes in the right half. The source node is allowed to transmit R times where R is the desired multicast rate.

We randomly generated networks having 20 and 40 nodes with different number of receivers and desired multicast rates as shown in Table 6.1. For each set of the parameters, we collected 100 random topologies whose multicast capacity for the chosen source and receiver nodes is the same as the desired multicast rate. That is, in each network tested, we tried to find a transmission scheme that achieves the maximum possible multicast rate while performing network coding only when required.

Table 6.1 shows the distribution of the calculated number of coding nodes for 100 random topologies for each parameter set. We notice that, though the multicast capacity can only be found assuming network coding first, to achieve the multicast capacity coding operation is often unnecessary at all. Even when network coding is needed, coding operations are required only at a small subset of the nodes. One may also observe the trend that more network coding becomes necessary as the network resources are more heavily used, i.e., the number of receivers and desired rate increase.

6.1.4 Location of Coding Nodes

Though it is hard to provide general guidelines regarding where to put the coding nodes because it is highly dependent on the topology and communication demands,

Table 6.1: Distribution of the calculated minimum number of coding nodes in 100 random topologies for each parameter set

Nodes	Receivers	Rate	No. of Coding Nodes					
			0	1	2	3	4	5
20	4	4	96	3	-	1	-	-
		6	97	3	-	-	-	-
	6	4	93	7	-	-	-	-
		6	95	5	-	-	-	-
	8	4	96	4	-	-	-	-
		6	91	7	2	-	-	-
40	4	4	82	17	1	-	-	-
		6	79	14	5	1	1	-
		8	86	11	2	1	-	-
	6	4	74	19	6	1	-	-
		6	66	22	7	4	1	-
		8	83	12	4	1	-	-
	8	4	68	26	5	1	-	-
		6	62	27	7	3	1	-
		8	65	25	7	2	-	1
	10	4	72	21	6	1	-	-
		6	57	30	5	5	2	1
		8	68	20	9	2	1	-

our algorithm allows us to do some interesting analyses that would reveal more insights on the problem. Let us first consider the flexibility of the location of coding nodes, which can be best illustrated by the sample networks used in Example 1 in Chapter 1.

In Fig. 1-1, both networks B and C require only one coding node to achieve the multicast of rate 2. In network B, network coding must be done exactly at node z to achieve the given rate, which we refer to as a *fixed* coding node. However, in network C, one coding node is needed, but it may not necessarily be node a because it can be node b instead, either of which we refer to as a *flexible* coding node.

Let us examine how many of the coding nodes we found in the experiments in Section 6.1.3 are fixed coding nodes. For each topology (that turns out to require at least one coding node), we now run our algorithm assuming that the above found coding nodes are treated as legacy nodes and at all other nodes network coding is free. That is, if some of the above found coding nodes are fixed, those nodes would

still remain as coding nodes despite the penalty imposed, and otherwise, other coding nodes would emerge instead. Table 6.2 shows the ratio of the fixed coding nodes to the whole number of coding nodes in 100 topologies generated in Section 6.1.3 for each parameter set. The ratio of fixed coding nodes shows a decreasing trend as the network resources are used more heavily, which is intuitive in the sense that heavy link usage may tend to create more alternative mixing opportunities among flows. Suppose that the coding nodes are randomly placed, which is likely to happen in mobile ad hoc networks. Our result suggests that as the network traffic becomes heavy, the location of the coding nodes may become more flexible, hence it is more likely that the randomly placed coding nodes may become useful.

Table 6.2: Ratio of the fixed coding nodes to the whole number of coding nodes found in 100 random topologies for each parameter set

Nodes	Receivers	Rate	Ratio of Fixed Coding Nodes
20	4	4	1.00
		6	1.00
	6	4	0.67
		6	0.83
	8	4	0.40
		6	0.44
40	4	4	0.29
		6	0.54
		8	0.40
	6	4	0.42
		6	0.30
		8	0.32
	8	4	0.38
		6	0.33
		8	0.43
	10	4	0.76
		6	0.40
		8	0.24

On the other hand, we may be able to place coding nodes in a more planned manner, for instance, when the given network is more static. Then, how can we find the good candidates for the location of the coding nodes? Suppose that we sampled some traffic patterns and found a number of common coding nodes. Are they likely to be useful for other traffic requests?

To answer this question, we picked two representative topologies, denoted by network $\mathcal{G}$ and $\mathcal{H}$, out of 100 random networks above with 40 nodes, 6 receivers and multicast rate 4. Network $\mathcal{G}$ requires 3 coding nodes ($\{15, 17, 19\}$) that are all fixed. However, network $\mathcal{H}$ needs 2 flexible coding nodes, hence we ran our algorithm 30 times to find 7 different coding node pairs involving a total of 7 nodes: $\{13, 18, 19, 24, 28, 32, 34\}$. Then, for both networks, we randomly chose 30 different sets of a source and 6 receivers such that multicast rate 4 is achievable, and ran our algorithm again (once for each source-receivers pair). For network $\mathcal{G}$, in 16 cases out of 30, network coding was found to be necessary with the number of coding nodes varying from 1 to 3 and the union of all 16 sets of coding nodes was $\{15, 17, 19, 21, 24, 26\}$. For network $\mathcal{H}$, one to three coding nodes were found to be necessary in 26 cases and the union was $\{13, 22, 24, 28, 32, 34\}$. It is interesting to note that in both networks $\mathcal{G}$ and $\mathcal{H}$, at least half of the coding nodes were in common with different communication demands. That is, if the latter experiments (sampling 30 different source-receivers pairs) had been performed first and the coding nodes had been placed accordingly at the union of the found locations, the original communication demands would have been met without requiring additional coding nodes (or emulated coding operations at legacy nodes). Though the experiment is based on a very limited number of cases, it may indicate that deploying coding nodes in the common locations for a number of sampled traffic patterns may actually work well in practice for small to moderate size wireless networks.

6.1.5 Interactions among Coding and Non-Coding Nodes

Our algorithms is based on GA which is a direct search method, hence at any stage of the algorithm a set of working network codes is available. In our algorithm's distributed setup, this means that each merging node stores N actual coding schemes, where N is the population size, and if the i-th (i=1, ..., N) chromosome has no or only a single 1, the corresponding coding scheme is routing. Note that, at the end of the algorithm, all the source node has to do is to transmit the index, say j, of the best chromosome (out of N chromosomes in the last population) with the data. Then,

each downstream merging node simply operates according to the j-th chromosome stored at the node, whether it is routing or coding.

Therefore, the interaction problem is already dealt with implicitly within the framework of our algorithm, whether the algorithm is used to find the potential location of the coding nodes or to minimize the number of legacy nodes that require coding while the locations of the coding nodes are already fixed.

6.1.6 Effect of Packet Erasures

Let us now evaluate the effect of the losses on the performance of the algorithm. The adverse effect of packet losses appears mainly in the form of delayed convergence of the algorithm, or failure to converge in case of excessively severe losses. Given that the main idea of the temporal distribution is to employ multiple subpopulations that are managed independently, we may intuitively expect that the negative effect of independent packet losses would be mitigated by employing the temporally distributed structure, which we will verify through simulations.

Let us denote by **"Algorithm S"** the spatially distributed algorithm as presented in Section 4.1 with the changes described in Section 6.1.2. For comparison, we let **"Algorithm G/M"** denote the algorithm with both spatial and temporal distribution as described in Section 4.2.2. For network $\mathcal{G}$ in Section 6.1.4, we compared the elapsed time, in terms of generations of GA, until the optimal solution 3 is found by Algorithms S and G/M. For Algorithm G/M, we let $k = 5$, i.e., there are 5 subpopulations of size 200, and migration frequency is set to 5 (generations). In fact, the longest path from the source to a receiver in network $\mathcal{G}$ is 22 hops, so if we assume that a single transmission takes a unit time, the end-to-end delay is at least $44(= 2l)$. For a completely pipelined operation we may set $k = 44$, which may result in more collisions in practice due to more frequent transmissions, but instead we experimented with a moderate value $k = 5$.

Due to the stochastic nature of GA, the number of generations until the desired solution is found varies for each run. We repeated the experiments 30 times for both algorithms with varying erasure rates from 0 to to 5% as shown in Table 6.3. We as-

sumed that erasures happen independently for each transmission both in the forward or backward evaluation phases and that the affected packets are simply dropped and never retransmitted. When erasures are introduced, the calculated fitness values are not guaranteed to be correct at each generation. We used a criterion such that the fitness value was assumed to be correct when it appeared the same in two consecutive generations, which happened very rarely if the fitness value was actually incorrect within the erasure levels we tested. For higher erasure rates, one may need to have a stricter criterion.

Table 6.3: Number of generations required to find the known optimal solution in the presence of packet erasures.

	Loss	Avg	Std	Min	Max
S	0	32.10	17.35	19	90
	1%	47.13	35.45	25	153
	2%	51.00	38.58	30	249
	3%	64.83	18.51	40	113
	4%	105.4	67.24	32	378
	5%	163.97	88.28	58	389
G/M	0	19.78	1.92	16	27
	1%	24.00	2.29	21	30
	2%	29.17	3.60	23	37
	3%	37.87	7.33	20	60
	4%	46.87	9.09	20	60
	5%	64.33	21.49	26	122

First of all, both algorithms were able to operate without much disruption up to 5% of erasure rate. In addition, we observe the general trend that the higher the erasure rate, the larger the number of generations required, as depicted in Fig. 6-1 where the range shows the standard deviation around the mean value.

Let us first look at the no-loss case. Algorithm G/M finds the (known) optimal solution roughly 1.62 times faster than Algorithm S on average. Much more remarkable is the difference in their standard deviations. Hence, if the optimal solution is not known and the algorithm is run until some fixed number of generations, which is the case in practice, the temporal distribution scheme may lead to better solutions much more reliably.

It is very interesting to note that the advantage of the temporal distribution

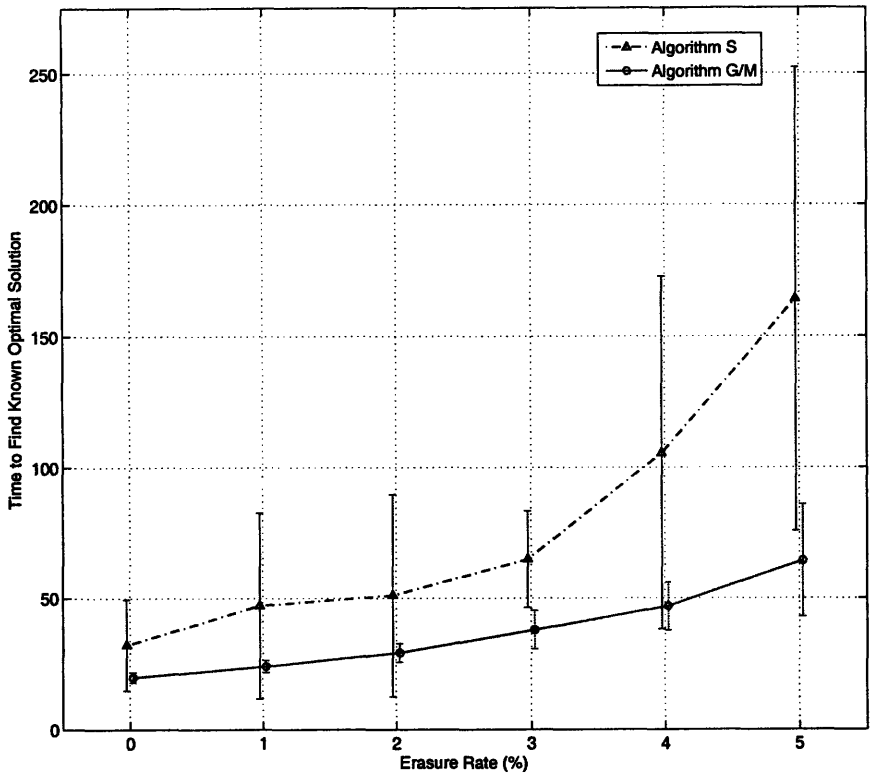


Figure 6-1: Mean and standard deviation of the time to find the optimal solution.

becomes more pronounced in terms of the mean and standard deviation of convergence time, when erasures are taken into account, especially at higher rates (see Fig. 6-1). From this observation, we can conclude that having multiple subpopulations that are subject to independent erasures and periodic migration (i.e., exchange of best solutions) offers a significant advantage in overcoming erasures, or more generally various kinds of losses in the network.

6.2 Fault-Tolerant Multicast in Optical Networks

Providing resilient service against failures is a crucial issue for today's optical networks because they operate at very high data rates and thus a single failure may cause a severe loss of data. A variety of protection techniques have been extensively studied for the fault-tolerant operation of optical networks of either ring (e.g., [21,57]) or mesh

topologies (e.g., [40, 52]). Among them, we particularly focus on the path protection scheme with live back-up, which provides extremely fast recovery as only the receiver nodes need to react after failures [43]. A conventional way to implement such a protection scheme is to transmit two live flows, a primary and a back-up, along link-disjoint paths so that upon link failure the receiver node can switch to the back-up flow. However, it may require an excessive amount of redundant capacity as back-up capacity is not shared among connections.

Regarding the role of network coding in protection or restoration in general, [26] presents an information-theoretic framework for network management for recovery from nonergodic link failures. Reference [30] presents the concept of 1+N protection that uses network coding over p-cycles, however, without consideration of the cost issue.

More recently, [45] demonstrates that network coding can lead to significant savings in the back-up resources for the multicast scenario protected against link failure by live back-up. Notably, in contrast with other network coding solutions with a generic network models, [45] takes into account a unique and crucial characteristic of optical networks: converting photonic streams into electronic signals for data processing and then back to photonic streams for retransmission, which is called Optical/Electronic/Optical (O/E/O) conversion, is an expensive procedure. Since arbitrary coding operations must be performed in the electronic domain, [45] restricts the coding operations only to bitwise XOR, which can be done within the optical domain using a photonic bitwise XOR hardware. However, from the perspective of network coding, such bitwise XOR coding is too restrictive given that most existing code construction methods require a large finite field.

In this section, we demonstrate how our evolutionary approach can be effectively applied in a number of different ways to better tackle the problem of fault-tolerance multicast in optical networks.

6.2.1 Problem Formulation and Assumptions

As before, the network is given by a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of directed links, but now we generalize our network model in two ways. First, each link now has an arbitrary positive integer capacity representing the maximum number of wavelengths carried on the link. Second, to represent a bidirectional link between two nodes, we now allow a pair of unidirectional links with opposite directions, each assumed to have a fixed capacity. Hence, now our network may abound with directed cycles. We assume that a single wavelength channel is the unit granularity of the optical transport and thus a photonic stream with the data rate of a wavelength is not split into smaller streams.

We again consider the single multicast scenario in which a single source $s \in V$ wishes to transmit the data that amount to R, a positive integer, wavelengths to a set $T \subset V$ of receiver nodes, where $|T| = d$. In addition, we require the *fault-tolerance* condition which implies that the given multicast rate R is still achievable in case of any single link failure by switching only at the affected receivers. Note that for a bidirectional link, which we represent by a pair of unidirectional links in each direction, a single link failure implies that the both unidirectional links fail. Though we primarily focus on the protection against a single link failure, we will show, as we develop our method, that it is easy to generalize our method to the case of multiple link failures.

One of our objectives is to minimize the link cost, which more precisely is the total number of wavelengths used to achieve a fault-tolerant multicast of rate R. However, as mentioned earlier, network coding at interior nodes requires O/E/O conversions, which may incur significant costs that may cancel out some savings in the link cost. Therefore, as the second objective, we need to minimize the coding cost, which is the number of wavelengths that require network coding.

Having more than one objective, an optimal decision needs to be made in a different way from the case of the single objective of minimizing network coding resources. In the next two subsections, we present two different strategies to handle this problem.

6.2.2 First Strategy: Two-Stage Method

Our first strategy is motivated by the following preliminary experiments, in which, though, the fault-tolerance requirement is not taken account:

Example 5 *Let us take two ISP topologies, ISP 1755 and 3967, from the Rocketfuel project [53], based on which we randomly select a source node and 10 receiver nodes and using the algorithm in [41] we calculate a subgraph that sets up a minimum-cost multicast of the given target rate, assuming network coding is performed everywhere. For each target multicast rate of 2, 3 and 4, we repeated the simulations 30 times on both topologies.*

Then, to each such minimum-cost subgraph, we applied our evolutionary approach to minimize the number of coding links required. Interestingly, in all such 180 subgraphs (30 for each target rate and each ISP topology), our algorithm found that the number of coding links required is zero. □.

The above example suggests that, while *assuming* network coding is necessary to calculate a minimum-cost subgraph, there may be very few links/nodes where network coding is actually required in the resulting subgraph.

Let us now describe how our two-stage method proceeds:

- In the first stage, we assume that network coding is allowed everywhere and calculate a subgraph, using traditional optimization methods, that requires the minimum link cost to set up a multicast connection that achieves the target rate and satisfies the fault-tolerance condition.

- In the second stage, based on the calculated subgraph, we apply our evolutionary approach to minimize the coding cost.

Note that, after the first stage, we are guaranteed to have the link cost minimized and in the second stage, as suggested by Example 5, it may be possible that in most cases network coding is found to be not required at all, even with the fault-tolerance requirement. If indeed network coding is not required, we end up reaching the ideal point where the found solution is optimal in the both objectives.

If, however, a nonzero number of wavelengths are found to require coding in the second stage, this two-stage method cannot provide further information on the possible tradeoff between the coding and link costs because the two optimization processes were performed separately. In such a case, our evolutionary approach with multiple objectives, described in Section 5, becomes useful, as we will describe in the next subsection.

Let us now provide more details of each stage of the two-stage method.

First Stage: Link Cost Optimization

At this stage, we do not restrict network coding at all. Then, from [1, Theorem 1], the necessary and sufficient condition for a feasible fault-tolerant multicast is that the max-flow between the source and each receiver remains at least R even if, for any single pair of nodes, the links between them all fail.

Consider the following optimization problem:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{(i,j)\in E} z_{(i,j)} \\
\text{subject to} \quad & c_{(i,j)} \geq z_{(i,j)}, && \forall (i,j) \in E, \\
& z_{(i,j)} \geq x^{(t)}_{(i,j)} \geq 0, && \forall (i,j) \in E, t \in T, \\
& y^{(t)} \geq x^{(t)}_{(i,j)} + x^{(t)}_{(j,i)}, && \forall \{(i,j) | (i,j) \in E \wedge (j,i) \in E\}, t \in T, \\
& y^{(t)} \geq x^{(t)}_{(i,j)}, && \forall \{(i,j) | (i,j) \in E \wedge (j,i) \notin E\}, t \in T, \\
& \sum_{\{j|(i,j)\in E\}} x^{(t)}_{(i,j)} - \sum_{\{k|(k,i)\in E\}} x^{(t)}_{(k,i)} = \begin{cases} R + y^{(t)} & \text{if } i = s, \\ -(R + y^{(t)}) & \text{if } i = t, \\ 0 & \text{otherwise,} \end{cases} && \forall i \in V, t \in T, \\
& c_{(\cdot)}, x^{(\cdot)}_{(\cdot)}, y^{(\cdot)}, z_{(\cdot)} : \text{integers}, && (6.1)
\end{aligned}
$$

where $C_{(i,j)}$ is the maximum number of wavelengths that can be transmitted on link (i,j).

Theorem 4 *The integer linear optimization problem* (6.1) *finds the minimum link cost for a multicast of rate R protected by live back-up against a single link failure.*

Proof: Note that this is a slight generalization of Theorem 1 of [41]. Suppose we have a feasible solution for the integer linear optimization problem (6.1). Then, let us consider a subgraph where the capacity of each link (i, j) is $z_{(i,j)}$, which represents the total number of wavelengths carried by link (i, j). For any $t \in T$, the maximum flow from s to t is $R + y^{(t)}$, where $y^{(t)}$ is the maximum amount of flow carried on a single link (or a pair of unidirectional links that represents a bidirectional link) between any pair of nodes, and thus it remains at least R in case of any single link failure. Hence, by Theorem 1 of [1], there exists a network code with flow $z_{(i,j)}$ on each link (i, j) that is feasible in case of any single link failure.

Conversely, let us assume that we have a fault-tolerant network code with integer flow $z_{(i,j)}$ on each link (i, j). Then, on the subgraph having $z_{(i,j)}$ as the capacity of each link (i, j), there must exist flow of size R from s to each $t \in T$, by Theorem 1 of [1], even in case of a failure on the link that carries the largest amount of flow, which is represented by $y^{(t)}$. Therefore, we have a feasible solution for the optimization problem (6.1). □.

The optimization problem (6.1) contains integer constraints, hence NP-complete in general, but it is of a fairly compact size as the numbers of variables and constraints grow only in the order of $O(|E||T|)$ and $O((|E| + |V|) \cdot |T|)$, respectively.

Second Stage: Coding Cost Optimization

Given that the optical networks are usually designed to abound with cycles and also that we allow bidirectional links, our network is very likely to have many directed cycles, even in the resulting subgraph after the first stage. Hence, our evolutionary approach may not work as presented in Chapters 3 and 4; more specifically, the parts that do not apply directly to cyclic networks are 1) the algebraic feasibility test and 2) the distributed implementations. We note that the distributed implementation is not an issue here because any live back-up protection scheme must work in a preplanned fashion and thus computations can be done centralized in the planning stage.

Recall that the algebraic feasibility test (presented in Sections 2.3 and 3.2), we explicitly deal with the scalar coefficients that appear in the system matrix assuming that the network operates with zero delay (and thus the network is cycle-free). In the presence of cycles, delay must be taken into account, hence the system matrix becomes a matrix over the polynomial ring with coefficients that are *rational functions* in the delay variable D [33]. In this case, the matrix computation involves calculating the coefficient for each power of the delay variable D, which in general may render the feasibility test prohibitively inefficient.

Note that, as shown in the proof of Theorem 2, the algebraic feasibility test can be interpreted in the graph theoretic framework, which works regardless whether the network is acyclic or not. To perform the graph theoretic feasibility test, let us first replace each link with capacity larger than 1 into multiple copies of unit-capacity links. Then, we decompose each node with more than one incoming link that is not a receiver in the same way as in the proof of Theorem 2 (see Fig. 3-3 for an example). (For a receiver node with a nonzero out-degree, introduce a virtual receiver and decompose the original receiver node as other merging node, after which we introduce a link from each incoming auxiliary node to the virtual receiver.) Then, there is a one-to-one correspondence between the set of binary variables in the chromosome and the set of the introduced links between auxiliary nodes.

Then, for the feasibility test of a chromosome, we first delete all the links introduced between auxiliary nodes that are associated with 0's in the chromosome. Then, we calculate the max-flows between the source and the receivers without consideration of the link cost, after which we verify the fault-tolerance requirement by taking out the link with the largest amount flow. Note that, unlike the algebraic case, this feasibility test incurs no error.

For a feasible chromosome, the number of coded wavelengths is the same as the number of coding links in the graph as we have decomposed all links into unit-capacity links. As before, the number of coding links can be calculated simply by counting the number of coding vectors with at least two 1's (or in the graph, the number of outgoing auxiliary nodes with tow or more remaining incoming links).

All other remaining parts of the GA proceed in exactly the same way as presented in Section 3, however, this time with the centralized computation.

6.2.3 Second Strategy: Multi-Objective GA

The second strategy utilizes our evolutionary approach with multiple objectives described in Section 5. However, there are two operational changes that need to be made here. First, the fitness evaluation is done using the graph decomposition method as in the second stage of the two-stage method presented above. This time, however, as the link usage must be controlled by the chromosome, we need to assign binary variables even to the outgoing links of a node with a single incoming link. Second, the whole algorithm operates in a centralized manner, hence it proceeds as shown in Figure 3-2 with the selection mechanism replaced with that described in Section 5.2.

6.2.4 Experimental Results

We experimented the above two strategies based on the NSFNET topology depicted in Figure 6-2 with 14 nodes and 21 links. We assume that all links in the network are bidirectional and have capacity, the maximum number of wavelengths that can be carried, 4 in *each* direction. Hence, a single link failure may disrupt up to 8 wavelengths transmitted in the both directions.

On this topology, we generated 100 random multicast scenarios, for each of which we randomly choose a source node out of the 5 nodes on the left-hand side (numbered from 1 to 5 in Figure 6-2) and three receiver nodes out of the remaining 9 nodes. We set the target multicast rate to 4, i.e., we wish to set up a multicast that delivers 4 wavelengths with the live back-up protection against any single failure on the bidirectional links.

First, we applied the two-stage method to each of 100 multicast scenarios. Table 6.4 summarizes the calculated minimum number coded wavelengths. Interestingly, we observe that, even with the fault-tolerance constraint, the minimum-cost multicast can mostly (in 99 out of 100 cases) be achieved without network coding at all, though

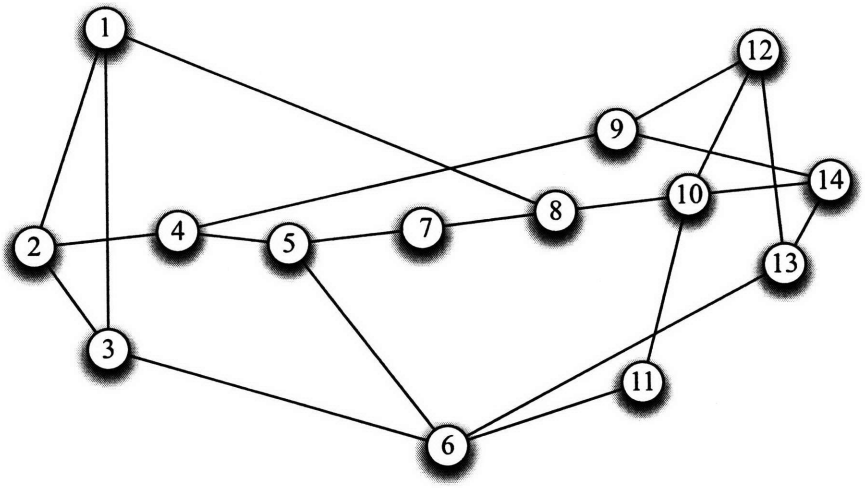


Figure 6-2: NSFNET topology.

we have assumed network coding when calculating the minimum-cost subgraph.

Table 6.4: Distribution of the calculated minimum number of coded wavelengths in 100 random multicast scenarios on NSFNET topology.

Number of Coded Wavelengths	0	1	≥ 2
Number of Scenarios	99	1	0

For the single case where network coding is found to be required to achieve the minimum link cost, we applied our second strategy, the multi-objective evolutionary approach. Figure 6-3 depicts the calculated non-domination front for the link cost and the number of coded wavelengths. Note that there are two non-dominated solutions: one yields the link cost of 38 while requiring one coded wavelength and the other achieves a slightly higher link cost of 40 but without network coding. The optimal decision depends on the relative costs of link usage and O/E/O conversions required for network coding; i.e., we may want to employ an O/E/O conversion for network coding at a node to save 2 wavelengths, or if the cost of an O/E/O conversion exceeds that of 2 wavelengths, we can eliminate the coding requirement altogether, spending extra 2 wavelengths.

The above simulations demonstrate that the two-stage method can often lead to the ideal point where we can achieve a minimum-cost fault-tolerant multicast without requiring network coding at all. In such cases, the benefit provided by network

coding is that a more tractable optimization problem, than the conventional methods involving Steiner trees, can be formulated by simply assuming network coding, rather than it is actually being used to save costs. The simulations also show that, when network coding actually needs to happen, our multi-objective evolutionary approach can lead to much more informed decisions on whether or where to employ network coding.

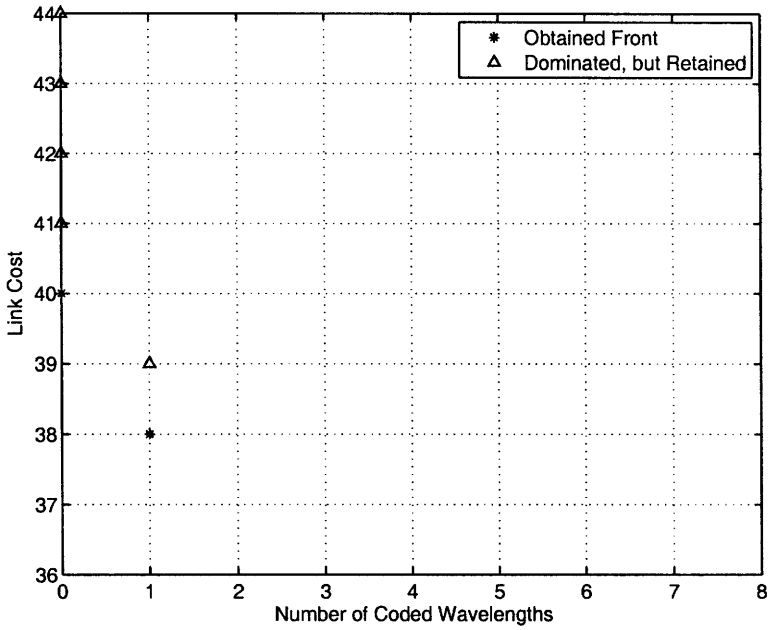


Figure 6-3: Calculated non-domination front for the case where a coded wavelength is required as shown in Table 6.4.

Chapter 7

Beyond Multicast

So far we have considered the single-session multicast scenario where all receivers demand the same source information. In more generalized networking scenarios where receivers may demand different sets of information out of multiple sessions, finding the optimal network coding strategy is still an open question. In this chapter, we investigate how our evolutionary framework can be generalized to fill the wide gap between the presumably difficult quest for optimal inter-session coding and many existing approaches that are constrained within very restrictive coding strategies.

7.1 Related Work

In the multicast scenario with a single session, the theory of network coding is well founded on the famous theorem by Ahlswede et al. [1] that characterizes the capacity region by the max-flow (min-cut) bounds. Subsequently, it is shown that the optimal capacity can be achieved using only scalar linear network codes [36].

However, when we generalize the problem such that there are more than one session and receivers may demand different sets of information, characterizing the capacity region becomes prohibitively difficult and even its inner/outer bounds cannot be computed in practice [6,59–61]. Moreover, linear coding is shown to be insufficient for optimal coding in the multi-session case [12]. Using connections between matroid theory and network coding, it is shown that Shannon-type information inequalities

are not sufficient for computing network coding capacities [13]. A graph theoretic approach is proposed as a systematic method for deciding solvability of a given network with either linear or nonlinear codes [54], whose scalability issue, however, is unresolved. Even within linear codes, the solvability decision problem is shown to involve Gröbner basis computation [33], whose complexity may prohibit practical implementations for large problems. Recently, [14] shows that it is possible to construct an associated network whose connection problem is solvable if and only if the set of polynomials is solvable, suggesting that solving a general network coding problem is equivalent in terms of complexity to solving a set of polynomial equations.

In short, it is very unlikely that a network coding strategy that is theoretically optimal will soon emerge. Nevertheless, there have been a number of studies that demonstrate various utilities of network coding in the multi-session scenario, based on some restricted class of codes in terms of the coding operations allowed and/or the location of decoding. Katti et al.'s opportunistic coding [31] leads to a restricted, yet very practical coding scheme, taking advantage of the broadcast nature of wireless medium. In their coding scheme, multiple, possibly more than two, packets can be combined using bit-wise XOR, but decoding needs to be done immediately at the neighbors of the coding node.

If we wish to perform decoding at receivers, we may need to put an even stronger restriction: XOR operations are allowed only between two flows, thus called *pairwise* XOR coding. The simplicity of coding and decoding operations of pairwise XOR codes allows the code construction problem to be described as flow formulations, which thus can be solved jointly with various other network flow problems. Traskov et al. [55] present linear and integer optimization formulations for the class of pairwise XOR coding for multiple unicast sessions. Ho et al. [23] develop back pressure algorithms for finding approximately throughput-optimal network codes within the class of pairwise XOR coding. Eryilmaz et al. [17] propose a dynamic scheduling strategy for routing and coding that supports any throughput strictly within the nontrivial region of achievable rates for multiple unicast sessions given by Traskov et al. [55].

Note that the benefit of pairwise XOR coding within that framework presented

in [55], in terms of savings in link cost compared with traditional routing, is found to be only modest depending on network topologies. It is not clear whether such a modest gain is because the simulations performed in [55] were restrictive, or it actually indicates the limitations of the pairwise XOR coding.

However, it is not easy to verify whether we can obtain an improved gain if we lift the restriction on the number of sessions that can be coded together or the type of coding operation. The flow formulations in [55] or [23] may be generalized to represent a binary XOR coding scheme that allows coding among more than two sessions. However, keeping track of all possible combinations of coded streams between up to k sessions would require at least $O(m^k|E||V|)$ variables, where m is the total number of sessions, which leads to a prohibitively large number of constraints, hindering practical implementations. Beyond the binary field, a path-based characterization is presented for the feasibility of the connection problem with two unicast sessions [56], based on which a distributed rate control algorithm is proposed [32]. However, a generalized characterization in the case of coding among more than two sessions still seems difficult.

For more general coding schemes beyond the pairwise binary XOR that allow decoding at receivers, a linear optimization problem is proposed by Lun et al. [41], whose minimum cost is shown to be no greater than the minimum cost of any routing solution. However, its formulation is based on the given set of the source processes that can be mixed on each link, which still remains difficult to decide optimally.

7.2 Our Coding Strategy

7.2.1 Problem Setup

As before, we consider the standard framework where the network is given by an acyclic directed multigraph $G = (V, E)$ where each link has a unit capacity and links with larger capacities are represented by multiple links. Only integer flows are allowed, hence there is either no flow or a unit rate of flow on each link.

Let us assume that there are R independent random processes of unit entropy rate, denoted by $\mathcal{X}=\{X_1, X_2, ..., X_R\}$, originating at $s(\geq 1)$ source nodes. There are d receiver nodes, $v_1, v_2, ..., v_d$. For each receiver v_i $(i = 1, ..., d)$, we denote the set of requested source processes by $\mathcal{X}_i \subset \mathcal{X}$. We assume that for each source process in $\mathcal{X}_i$ $(i = 1, ..., d)$, there exists a path from its originating node to receiver v_i; otherwise, it is easy to check and declare that the problem is not solvable. Other than this connectivity condition, we do not put any restriction on the source processes each receiver node may request, i.e., $\mathcal{X}_i$ can be any nonempty subset of $\mathcal{X}$ provided that receive v_i is reachable from the originating nodes of the contained source processes.

For this generalized scenario, there is no simple characterization known for the feasibility of the connection problem with network coding, even for the simpler case of unicast connections, i.e., when all $\mathcal{X}_i$'s are disjoint [33, 37]. Though linear coding has been shown to be suboptimal in general [12], here we focus on the scalar linear coding in a general form, i.e., we do not restrict ourselves within either binary XOR operation or pairwise mixing.

Because the feasibility characterization still remains hard even within linear coding [33], we assume that the capacity constraints can be relaxed for some links, i.e., links can be scaled up, if necessary, to allow multiple transmissions. Note that this may be the case in many practical networking scenarios; e.g., in wireless networks, nodes can exploit more capacities at the expense of more energy, and in optical networks, capacity itself is rarely a limited resource, though using more resources incurs an additional cost. With this assumption, the feasibility problem can be resolved by scaling links appropriately.

Then, we mainly focus on finding a cost-efficient transmission scheme using network coding. Hence, our primary objective is to minimize the link cost required to satisfy the given communication demands. Later, we consider another objective of minimizing the coding resources, i.e., number of coding nodes/links.

7.2.2 Background

For the multicast case, the solvability of a connection problem boils down to whether the max-flows between the source node and each of the receiver nodes all exceed the desired multicast rate [1], which then can be translated into the algebraic framework such that the transfer matrices relating the input and output processes are all nonsingular [33]. Note that the determinant of each of the transfer matrices can be represented as a polynomial with the link coefficients as variables. If we let $\underline{\xi}$ denote the vector consisting of all the link coefficients, the problem of finding a feasible network code becomes finding an assignment of number to variables $\underline{\xi}$ such that the product of the determinant polynomials does not evaluate to zero, which can be done relatively easily with many efficient randomized algorithms, e.g., as in [27].

As we go beyond the multicast case, we now have another condition for solvability: within the transfer matrix, the submatrices relating the input processes that are not requested to the corresponding output processes at the receiver nodes should evaluate to zero (the first condition in Theorem 6 in [33]). Let us denote the *entries* of the submatrices that have to evaluate to zero by $f_1(\underline{\xi}), f_2(\underline{\xi}), ..., f_K(\underline{\xi})$, which we refer to as *interference polynomials*. As in the multicast case, we still have the condition that the submatrices associated with the input and output processes specified in the connection requests should be nonsingular (the second condition in Theorem 6 in [33]). Let $g_1(\underline{\xi}), g_2(\underline{\xi}), ..., g_L(\underline{\xi})$, referred to as *determinant polynomials*, be the determinants of the submatrices that should be nonsingular. Then, to obtain a feasible network code, we need to find an assignment of numbers to variables $\underline{\xi}$ that render all interference polynomials zero and all determinant polynomials nonzero.

While finding an assignment that makes determinant polynomials nonzero can be done easily, the difficult part is that we have to solve the set of polynomial equations to find an assignment of numbers to $\underline{\xi}$ that makes them all zero, which we cannot expect to be done randomly.

To determine whether such an assignment is possible, [33] constructs an ideal generated by the interference polynomials and a function of the determinant polynomials,

and shows that the problem is equivalent to the problem of deciding whether the variety of the constructed ideal is empty. As a method to determine the emptiness of a variety, [33] suggests Gröbner basis computation. However, the worst case complexity of the Buchberger algorithm, which is used predominantly to compute a Gröbner basis, is doubly exponential [46]. Though [2] claims that such worst case might rarely happen, in our case the input to the Buchberger algorithm, i.e., the interference and determinant polynomials, may already have an exponential number of terms as each component of the transfer matrices involves a number of variables that scales up to the maximum number of ways to travel from a link to another in the network. Hence, calculating a Gröbner basis may not be a practical solution anyway.

Moreover, as solving a general linear network coding problem is no easier than solving a set of polynomial equations [14], it is unlikely that one can develop a specialized method that possibly provides more efficiency, utilizing some structural properties of the interference and determinant polynomials.

7.2.3 Selective Random Linear Coding Strategy

Our coding strategy is essentially the same as that we used for the feasibility test in the coding resource optimization problem for multicast. That is, we control the mixture of information at each node by deciding whether to include the input from a particular incoming link when calculating the output on each outgoing link. However, once the decisions are made, we calculate the output by forming a random linear combination of the inputs allowed. Hence, the name *selective random linear coding.*

Intuitively, the strategy we employ to deal with the interference polynomials is 1) to make some interference polynomials identically zero by zeroing out an enough number of associated components of $\underline{\xi}$ and 2) for the interference polynomials that remain nonzero, to allow enough degrees of freedom at the receivers so that the unwanted information can be successfully canceled out.

For a node v with d_{in} incoming links and d_{out} outgoing links, we assign a binary variable a_{ij} to each pair of the $i \in \{1, ..., d_{in}\}$-th incoming link and the $j \in \{1, ..., d_{out}\}$-th outgoing link. For the j-th ($j = 1, ..., d_{out}$) outgoing link, we refer to the associated

binary variables $a_j = (a_{ij})_{i\in\{1,\ldots,d_{in}\}}$ as a *coding vector* (see Fig. 3-1 in Section 3.1 for an example). The coding vectors are the variables that we need to decide in our coding strategy.

Once the coding vectors are given, we employ random linear coding at interior nodes, selectively using only those inputs associated with 1's in the coding vectors. More specifically, node v calculates the output y_j ($j = 1, ..., d_{out}$) on the j-th outgoing link as follows. Define the set I_j of indices as

$$I_j = \{1 \leq i \leq d_{in} | \text{ the } i\text{-th component of } a_j \text{ is } 1\}. \tag{7.1}$$

If we denote the input from the i-th ($i = 1, ..., d_{in}$) incoming link by x_i,

$$y_j = \sum_{i\in I_j} \text{rand}(\mathbb{F}_q) \cdot x_i \tag{7.2}$$

where $\text{rand}(\mathbb{F}_q)$ denotes a nonzero random element from $\mathbb{F}_q$. If the set I_j is empty, y_j is assumed to be zero.

Given a set of coding vectors, the feasibility verification can be done by first performing selective random linear coding at interior nodes as described above. Then, each receiver node v_i ($i = 1, ..., d$) performs Gaussian elimination to determine whether all the desired input processes can be recovered, with the interference part canceled out.

Again, this randomized decision rule incurs an error when nonzero polynomials evaluate to zero when randomly assigning values to variables; for zero polynomials, random assignments in the evaluation process do not affect the final result. Hence, the error probability is bounded by $1-(1-d/q)^{\nu}$ where q is the size of the finite field used for coding and ν is the maximum number of links in any set of links constituting a flow solution from the source to any receiver [27]. Note that this bound remains the same even if we scale up some of the links.

Note that we have to deal with an exponential number of possible assignments of the coding vectors, which again we address using a GA-based search method as

described below.

7.3 Computational Aspects

Our approach again is based on simple GA (Fig. 2-2). We first describe how the computational components of the approach need to be designed, and then in the next section, present how our approach can be distributed over the network.

7.3.1 Chromosome and Fitness Function

The decision variables in our coding scheme are a set of coding vectors as defined above in Section 7.2.3. Hence, each chromosome consists of the collection of all coding vectors associated with the nodes in V.

Recall that for the multicast case (Section 3.5), we could reduce the size of the search space by adopting the vector-wise representation where all coding vectors with two or more 1's were treated as a single vector of all 1's to represent the coded transmission. In this general problem, however, each different coding vector functions differently in terms of the interference, which may eventually affect the feasibility, and thus we cannot do the same truncation.

For a given chromosome $\underline{y}$, we must verify its feasibility first by performing selective random linear coding as described in Section 7.2.3. Once the feasibility test is done, given that our primary objective is to obtain the cost-efficient transmission scheme via network coding, the fitness function F is defined as

$$F(\underline{y}) = \begin{cases} \text{total cost of link usage}, & \text{if } \underline{y} \text{ is feasible,} \\ \infty, & \text{if } \underline{y} \text{ is infeasible.} \end{cases} \tag{7.3}$$

Note that if we wish to minimize the resources engaged in network coding, we may replace the link cost by the number of coding nodes/links for feasible chromosomes. Moreover, the two objective values, i.e., the link and coding costs, can be jointly considered to obtain Pareto-optimal solutions as in Chapter 5, which can provide

useful insights on the tradeoff between those two objectives.

7.3.2 Initial Population Construction

Typically the initial population of a GA is composed of random chromosomes. However, as pointed out in Section 3.3, inserting some pre-calculated chromosomes can greatly improve the performance of the algorithm. Though we may use a GA to find a feasible solution starting from a purely random initial population, such a process may be very slow due to insufficient information (and the difficulty in finding an appropriate one) in the fitness function to guide the initial search process toward a feasible solution.

Note that, from the assumption that we may relax the capacity constraint when needed, it is easy to construct a feasible solution by an appropriate link scaling; i.e., if we scale up the links with a sufficiently large factor, we can always obtain a feasible solution, though it may incur an excessive amount of link cost.

However, when we add a number of feasible solutions to the randomly generated initial population, care must be taken to insert only *neutral* feasible solutions in the sense that they are not particularly close to some local optimum. Otherwise, those inserted starting points tend to take over the whole population in the early stage of the evolution process so that the algorithm may end up converging to some neighborhood of the starting points. Recall that, for the network coding resource optimization problem for multicast, we added the all-one chromosome to the initial population, which is feasible by assumption but worst in terms of the coding cost because it corresponds to the case where network coding is performed everywhere.

Then what would be a good neutral starting point in our problem? For instance, the best routing solution or a solution after greedy removal of links may not be a good candidate; it is very unlikely to obtain a better feasible solution from such a local optimal solution by exchanging its subcomponents with another random solution and moreover, before such an unlikely event may happen, the added local optimal solution is still better than other random solutions in the population, thus it tends to be selected repeatedly, eventually dominating the whole population.

We create a neutral starting point by scaling up the links in G to make the problem multicast-like in the sense that each receiver node now receives all source processes that have a directed path to the receiver and then employing an all-one vector that indicates mixing everything at all interior nodes. Before proceeding, for each receiver node $v_i (i = 1, ..., d)$ in G, we let $\mathcal{Y}_i \subset \mathcal{X}$ be the set of the source processes from whose originating node, receiver v_i is reachable (i.e., there exists a directed path from the source process's originating node to node v_i). Also, we let $r_i (i = 1, ..., d)$ denote the size of set $\mathcal{Y}_i$. Now let us create an auxiliary network H from G by introducing a virtual source node S and adding a unit-capacity link from node S to the source node at which each source process $X_j \in \mathcal{X} (j = 1, ..., R)$ originates. Then, we let $f_i (i = 1, ..., d)$ be the value of the maximum flow from the virtual source node S to each receiver $v_i (i = 1, ..., d)$. We define k as

$$k = \left\lceil \max_{i \in \{1,...,d\}} \left(\frac{r_i}{f_i} \right) \right\rceil. \tag{7.4}$$

Theorem 5 *Let G' be a scaled version of G with each link replaced by k multiple links, where k is defined as in (7.4). In network G', a network code that performs random linear coding using all available inputs at every interior node in a sufficiently large finite field is feasible.*

Proof: Let us create another auxiliary network H' by scaling up the links in H by a factor of k, except those added later between the virtual source S and the original source nodes. For each receiver $v_i (i = 1, ..., d)$, let f_i' denote the max-flow from node S to node v_i. Then, as the links are scaled up,

$$f_i' = \min[k f_i, r_i], \tag{7.5}$$

where f_i is the corresponding max-flow in network H. However, by definition of k, $k \geq \frac{r_i}{f_i}$, and we can conclude

$$f_i' = r_i. \tag{7.6}$$

Now, for each receiver node $v_i (i = 1, ..., d)$, we add a unit-capacity link directly

from the original source node of each source process in $\mathcal{X} \setminus \mathcal{Y}_i$. With those $(R - r_i)$ direct links added, the max-flow from node S to node v_i becomes R, for each $1 \leq i \leq d$. Thus, we can regard the problem as a multicast scenario in network H' and, by employing random linear coding at every interior node in a sufficiently large finite field, each receiver node v_i can decode all R source processes.

Now, let us take any such multicast network code, denoted by $\mathcal{C}$, for network H'. We can relocate R source processes at the virtual source node S to the original source nodes, by treating each linearly combined version of the R source processes as a new source process.

For a particular receiver node $v_i (i = 1, ..., d)$, suppose it has $d_i n$ incoming links including the $(R - r_i)$ direct links from the source nodes. The received data, represented by d_{in} row vectors each of length R, will form a matrix of full rank R. Note that the $(R - r_i)$ source processes in $\mathcal{X} \setminus \mathcal{Y}_i$ are carried only by those $(R - r_i)$ direct links, hence the row vectors corresponding to those $(R - r_i)$ direct links are linearly independent with the remaining $d_{in} - (R - r_i)$ rows. If we remove the $(R - r_i)$ columns corresponding to the source processes in $\mathcal{X} \setminus \mathcal{Y}_i$, the remaining $d_{in} - (R - r_i)$ rows form a matrix with full rank r_i. Therefore, without the added direct links, all source processes in $\mathcal{Y}_i$, hence those in $\mathcal{X}_i \subset \mathcal{Y}_i$, can be recovered.

Therefore, network code $\mathcal{C}$ is also feasible in network G', which corresponds to network H' without the virtual source S and the direct links from the source nodes to receiver nodes. □

Hence, we first create network G' by scaling links in G by a factor of k and then use the all-one chromosome as a neutral starting point in the initial population. Note that the all-one chromosome is feasible but has the worst cost in terms of either the link cost or the coding cost. Therefore, it may not bias the initial population toward any particular suboptimal solution.

7.3.3 Genetic Operators

Recall that in the multicast case, the vector-wise genetic operators have led to a significant performance gain compared with the conventional bit-wise genetic operators,

exploiting the modularity among variables when interchanging or perturbing subcomponents of chromosomes. Similarly, we can define the vector-wise genetic operators for this general problem as follows:

- **Vector-wise uniform crossover:** We let two chromosomes subject to crossover exchange each full coding vector independently with the given crossover probability.

- **Vector-wise mutation:** For each chromosome, we randomly regenerate each coding vector independently with the given mutation probability.

Note that in the multicast case, the superior performance obtained by the vector-wise genetic operators was in fact the result of the two combined effects: the exploitation of coding vector level modularity and the reduced search space size. This time, as mentioned earlier in Section 7.3.1, there is no space size reduction and thus we can only expect the effect of the modularity exploitation, which, in the multicast case (Section 3.7.3), is claimed to be the main source of the superior performance.

Given the positive effect of enforcing modularity in genetic operations, we may develop another set of genetic operators that further exploit, this time, the node-wise modularity. Note that each coding vector is a set of the bit strings designated for calculating the output on a single outgoing link, and each node has as many coding vectors as the number of its outgoing links. Let us define the node-wise genetic operators as follows:

- **Node-wise uniform crossover:** We let two chromosomes subject to crossover exchange the coding vectors associated with the outgoing links of each node with the given crossover probability.

- **Node-wise mutation:** For each chromosome, we randomly regenerate the coding vectors associated with the outgoing links of each node independently with the given mutation probability.

The intuition behind the node-wise genetic operators is that the coefficients of the links directly connected with each other or within a few hops away are more likely

to have strong dependencies than those associated with the links far away from each other. Note, however, that the node-wise genetic operators enforce a broader level of modularity than the vector-wise operators, exchanging or perturbing a larger number of coding vectors at once. If this is the right level of modularity, it would translate into faster convergence of the algorithm to the same or a better solution; otherwise, the algorithm may tend to converge prematurely to a lower quality solution. We will verify the effect of different genetic operators below through simulations.

7.4 Distributed Implementation of Algorithm

In the multicast case, we have shown that our evolutionary algorithm for the coding resource optimization can be implemented in a distributed manner and the information required for the coordination of the interior nodes can be calculated at the source node after receiving fitness vectors and transmitted in the form of a *coordination vector* at the renewal of the subsequent generation.

From the implementation perspective, most part of the algorithm remains the same as we generalize our algorithm for selective random linear coding. The most notable difference is that now there are multiple source nodes. The backward evaluation phase, which in the multicast case collects at the source node the feedback data after random linear coding is performed at the interior nodes, would now allow each source node to have only partial information about the fitness values of the chromosomes. Hence, we need some mechanism for the multiple source nodes to gather such partial information to calculate the full fitness values and also to share the resulting coordination vector.

7.4.1 Assumptions

Note that when we consider network coding among multiple sessions that may originate from multiple source nodes, it must be assumed in the first place that the source nodes can communicate with one another to find out the total number of the source processes being considered together for possible coding so that the coefficients for

linear coding can be aligned consistently across all the source nodes involved.

Hence, we assume that among the participating source nodes, we can pick one node that has a path to all other source nodes that allows communication in both directions and designate the node as the *master node* which serves as the main controller of the algorithm by gathering information from the source nodes and sending the calculated coordination vector to other source nodes. The detailed role of the master node will be described below.

Let us first assume that the links are already scaled with factor k as defined in (7.4). Later, in Section 7.4.3, we will discuss how the scaling can be done also in a distributed manner. In the description of the algorithm below, we assume that the fitness function is defined as in (7.3).

7.4.2 Details of Algorithm

The overall flow of our distributed algorithm is shown in Fig. 7-1 with the location of each procedure specified.

```
[S1]  initialize; (all nodes)
[S2]  run forward evaluation phase; (all nodes)
[S3]  run backward evaluation phase; (all nodes)
[S4]  send partial fitness to master node; (source nodes)
[S5]  calculate fitness; (master node)
[S6]  while termination criterion not reached (master node)
      {
[S7]    calculate coordination vector; (master node)
[S8]    fetch coordination vector from master node; (source nodes)
[S9]    run forward evaluation phase; (all nodes)
[S10]   perform selection, crossover, mutation; (interior nodes)
[S11]   run backward evaluation phase; (all nodes)
[S12]   send partial fitness to master node; (source nodes)
[S13]   calculate fitness; (master node)
      }
```

Figure 7-1: Flow of distributed algorithm for selective random linear coding

Initialization [S1]

The master node initiates the algorithm by transmitting to other source nodes a packet containing the following predetermined parameters: the total number of source processes and the ordering of those processes, population size N, the size q of the finite field to be used, crossover probability, and mutation rate. Each source node than passes the initialization packet to its downstream nodes so that all participating nodes can share the information.

Each interior node with d_{in} incoming and d_{out} outgoing links has to manage d_{out} coding vectors of length d_{in} to indicate the link states for a single chromosome; i.e., for population size N, each node must have $N \cdot d_{out}$ coding vectors to determine the operations at that node. For initialization, each node generates $N \cdot d_{out}$ coding vectors randomly and independently, while setting the first one to an all-one vector.

Forward Evaluation Phase [S2, S9]

Forward evaluation phase proceeds the same way as **[D2, D7]** in Section 4.1.2.

Backward Evaluation Phase [S3, S11]

Backward evaluation phase is mostly the same as **[D3, D9]** in Section 4.1.2 except that now we consider the link cost. Hence, the fitness vector has N components, whose i-th component conveys the link cost up to the location where the fitness vector is generated. Each component corresponding to an infeasible chromosome is signified by the infinite link cost.

Collection of Partial Fitness Values [S4, S12]

After the backward evaluation phases, now each source node has some partial information about the fitness values of N chromosomes. To complete the fitness evaluation, each source node send out its fitness vector to the master node, for which process we assume there exists a means of reliable communication.

Fitness Value Calculation [S5, S13]

Based on the fitness vectors received from other source nodes, the master node calculates the fitness values of N chromosomes simply by performing component-wise summation. Again, if an infinity were generated by *any* of the receivers, it should dominate the summations all the way up to its associated source, and thus the master node can calculate the correct fitness value for infeasible chromosomes.

Termination Criterion [S6]

We use the same termination criterion as in **[D5]** in Section 4.1.2.

Coordination Vector Calculation and Distribution [S7] [S8]

The master node calculate the coordination vector the same way as in **[D6]** in Section 4.1.2. Then, the calculated coordination vector is sent to all other source nodes through the assumed communication channel among the source nodes. Then, the local source nodes transmit the received coordination vector together with the pilot vectors in the next forward evaluation phase.

Genetic Operations [S10]

Based on the received coordination vector, each node can locally perform genetic operations and renew its portion of the population as described in in **[D8]** in Section 4.1.2. Note that the node-wise genetic operators also involve only locally available coding vectors, so the genetic operations can proceed the same way as before only with the different level of modularity.

7.4.3 Distributed Link Scaling

The scaling factor k, which is defined as in (7.4), can be found in a distributed fashion using a part of the algorithm described above. Note that k is the smallest integer scaling factor that makes the network code that mixes everything at every node feasible. Hence, we can find k by repeating the fitness evaluation process (procedures

[S2] through [S5] in Figure 7-1) up to R times with slight changes as follows: Each time we test scaling factor j ($j = 1, ..., R$) by calculating the fitness value of the population consisting of a single chromosome that has only all-one coding vectors. For each j, we let each node that originally has d_{in} incoming and d_{out} outgoing links operate as if it has $j \cdot d_{in}$ incoming and $j \cdot d_{out}$ outgoing links; i.e., each node transmits j times for each of its outgoing links. At the end of each fitness evaluation process (procedure [S5]), the master node can determine whether the current scaling factor j makes the all-one chromosome feasible; if so, the iteration stops and let $k = j$, and otherwise, continue the iteration with $j = j + 1$. Note that the iteration stops at or before $j = R$ because once we scale all links by R, the all-one chromosome becomes feasible as the max-flow from the virtual source S (as defined in Section 7.3.2) to each receiver v_i ($i = 1, ..., d$) is clearly at least r_i (as defined in Section 7.3.2). Once a correct scaling factor is found, the master node sends a signal to all nodes so that they can operate with the found scaling factor afterwards.

7.5 Performance Evaluation

In order to evaluate the performance of our algorithm in comparison with others', and also to compare the effects of different genetic operators, we first perform simulations for the multiple-unicast scenario, for which other network coding schemes are available. Then, to demonstrate our algorithm's ability to handle more general problems, we further consider the case of wireless networks with no restriction on the type of connection requests.

For the GA parameters, we set the population size to 2, which gave much superior performance in our preliminary experiments than the typical value of 10 used in other simulations. Other than the population size, we use the same parameters suggested in Section 3.6.

7.5.1 Multiple Unicast Connections

To measure the performance of our algorithm with different combinations of parameters, we performed a number of simulations for the multiple-unicast case, i.e., all connection requests $\mathcal{X}_i$'s ($i = 1, ..., d$) are disjoint. Our simulations are based on the grid network introduced in [55] (network D in Fig. 7-2). In [55], the cost of each link is assigned randomly, which allows only slim chances that the network coding advantage exists. Note, however, that network coding gain takes effect only if there exists an expensive bottleneck link that has to be used by a number of flows and also lower cost detours around it, which happens rarely when the connection requests and link costs are randomly chosen. Hence, in our experiments, we pick a cost assignment as depicted in Figure 7-2, where the links with a cost higher than 1 are highlighted by a thicker arrows with the actual cost shown by the side, to make the network coding advantage clearly exist at least for some connection requests.

For comparison, we take Traskov et al.'s pairwise binary XOR coding scheme [55]. We do not include the approach by Wang et al. [56] here because it can be shown that the grail structure considered therein does not lead to link cost savings in our problem setup; nevertheless, it does increase the capacity region when the capacity constraint is strictly enforced.

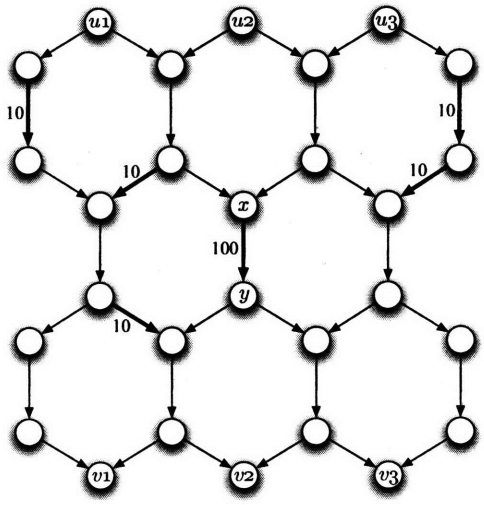


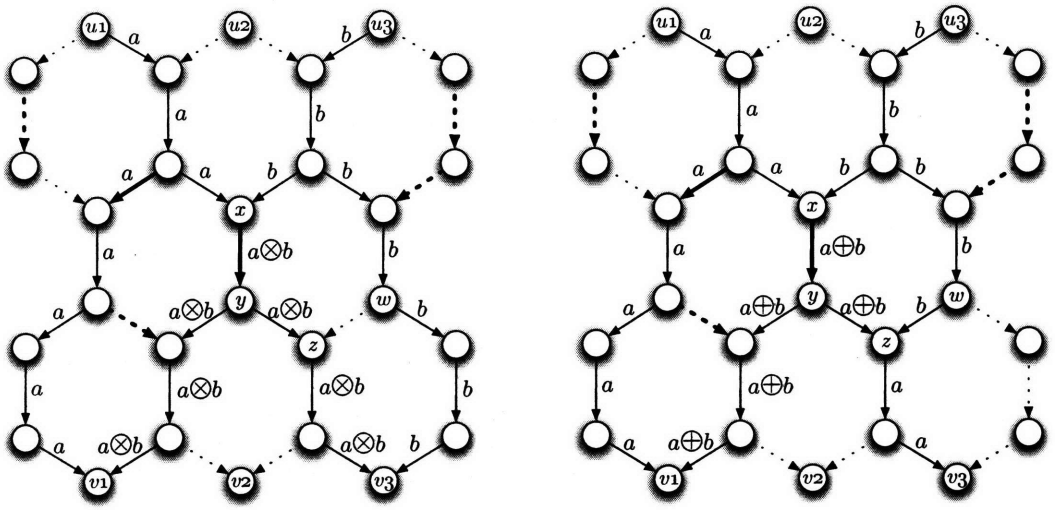
Figure 7-2: Grid network D for multiple-unicast simulations

Two-Connection Case

First, we consider the case of two connections, i.e., $R = 2$. We repeatedly ran our algorithm, varying the location of the source processes and connection requests such that two source processes X_1 and X_2 may originate at any of nodes $\{u_1, u_2, u_3\}$ and each source process may be requested at any of receiver nodes $\{v_1, v_2, v_3\}$. For comparison with the routing case and the pairwise binary XOR coding scheme, we used the multi-commodity formulation with integer constraints and Traskov et al.'s formulation [55], respectively.

Among all possible arrangements of the source processes and connection requests, there is only one case in which network coding saves the link cost: X_1 and X_2 originate at nodes u_1 and u_3, respectively, while X_1 and X_2 are requested at nodes v_3 and v_1, respectively.

Fig. 7-3(a) shows the best solution obtained by our selective random linear coding which requires the link cost of 131. Note that, for this simple problem, our evolutionary algorithm, despite its stochastic nature, always yields the same solution regardless of the genetic operators used. In comparison, the optimal cost achieved by routing is 212. In Fig. 7-3(a), we use the symbol $\otimes$ to represent linear combination with random coefficients from the designated finite field; i.e., $x_1 \otimes x_2 \otimes ... \otimes x_n = \sum_{i=1}^{n} \text{rand}(\mathbb{F}_q) \cdot x_i$, where each $\text{rand}(\mathbb{F}_q)$ represents a nonzero random element from the designated finite field $\mathbb{F}_q$. Interestingly, this example highlights the difference between our coding scheme and the pairwise binary XOR coding scheme, whose best solution, as depicted in Fig. 7-3(b), offers an even lower cost of 129. In our selective random linear coding, coding operation is performed only once at node x and decoding is done at the receiver nodes. On the other hand, in the best pairwise binary XOR code, another XOR operation is done at node z, which, in fact, serves as decoding at an interior node and consequently saves the link cost of 2 (by not sending β at node w along a longer path down to node u). Note that the same transmission strategy would not work for our selective random linear coding because another coding operation at node z would yield another random linear combination $(a \otimes b) \otimes b$ rather than a.



(a) Selective random linear coding

(b) Pairwise binary XOR coding

Figure 7-3: Comparison of best coding solutions obtained.

From this example, we observe that, while the both coding schemes provide advantages over traditional routing, our selective random linear coding scheme may require some additional link cost compared with the pairwise binary XOR coding, which can be considered the price of employing randomized coding in a finite field larger than the binary field. Note, however, that as we increase the number of connections, our coding scheme leads to a much more practical solution despite the possible expense of such additional link cost, as will be discussed next.

Five-Connection Case

Let us now increase the number of source processes to 5 and pick up an arrangement of the source processes and connection requests that allows the network coding advantage: $\{$(source process, source node, receiver node)$\} = \{(X_1, u_1, v_3)$, (X_2, u_1, v_1), (X_3, u_3, v_1), (X_4, u_3, v_2), $(X_5, u_3, v_2)\}$.

As opposed to the simpler problem in the previous subsection, the performance of our algorithm varies depending on the type of genetic operators used. Table 7.1 summarizes the performance of our algorithm with different genetic operators. For each type of genetic operators, we performed 30 simulation runs and at the end of each run we pick the best coding solution out of the last population. The first column

shows the lowest cost of those 30 best coding solutions and the next two columns show the average and standard deviation of the cost of those 30 solutions. For comparison, the best routing solution yields the optimal cost of 242.

The fourth column of the table displays the ratio of the algorithm runs, out of the total of 30, in which the cost of the best coding solution found is actually lower than the best routing solution. The fifth column calculates the average number of generations required for a coding solution that outperforms the routing solution, if found, to appear in the population. The last column shows the ratio of the algorithm runs in which a coding solution outperforming the routing solution is found before the 100-th generation, relative to the total number of algorithm runs where such a coding solution is ever found.

Table 7.1: Summary of the link costs of the coding solutions found by different genetic operators and related statistics. The best routing solution requires the link cost of 242.

	Link Cost			Outperform Routing		
	Best	Avg	Std	Ratio	At	<100
Bit-wise	181	204.2	10.52	1.00	296.7	0.07
Vector-wise	156	164.8	5.87	1.00	177.3	0.73
Node-wise	158	236.2	41.96	0.43	124.8	0.77

From Table 7.1, we first notice that our algorithm with the bit-wise or vector-wise operators reliably yields a network coding solution that outperforms the best routing solution. Between the two kinds of operators, the vector-wise operators lead to much better solutions, both in terms of the mean and standard deviation. Also, the number of generations required to find a network coding better than the best routing solution is much smaller on average for the vector-wise operators; most of the time it is found before the 100-th generation. Hence, we may conclude that the vector-wise operators allow the algorithm to find much better solutions much faster than the bit-wise operators.

For the case of the node-wise operators, our algorithm finds a network coding solution that exceeds the routing solution only in 43% of the simulations. However, in those successful simulations, such network coding solution is found much faster

than the case of the vector-wise operators. Hence, we may conclude that with the node-wise operators the algorithm tends to converge faster, but prematurely to a lower quality solution. This may be due to that the node-level modularity enforced by the node-wise operators is too strong in the sense that it changes too many coding vectors at once, which conceptually may correspond to setting the step too large in an iterative optimization scenario.

On the other hand, the optimization formulations for the pairwise binary XOR coding [55] fail to converge within a reasonable amount of time (during a full week of simulations) based on the simulation environment used in [55]. Note that the linear and integer programs in [55], even for this five-connection problem, contain around 68700 and 1400 variables (including the slack variables to handle the max operator in the constraints) and 67500 and 1700 constraints, respectively. Though we may have been able to obtain converged results if we had experimented it with a much faster machine, the point we would like to make here is that the optimization formulations considered in [55] may not provide a practical scalability as the number of connections increases. For comparison, our algorithm takes about 1.5 seconds for each generation in the same simulation environment, where we simulated each node's operation sequentially (one at each time from upstream to downstream nodes) using MATLAB on a single-processor machine. In a real network, where each node uses its own computational resources, we may expect much faster execution of our algorithm.

7.5.2 General Connections

We performed another set of experiments in fully generalized networking scenarios where the receiver nodes may request any combination of the available source processes. For this, we generated 100 random wireless networks where 40 nodes are placed randomly within a 10×10 square with radius of connectivity 3. A unit-rate hyperlink is originated from each node toward the set of nodes that are within the connectivity range and have a higher horizontal coordinate. In each random topology, 5 source processes were randomly placed at 5 nodes chosen in the left half and each of 5 receiver nodes randomly chosen in the right half demands a randomly chosen

nonempty subset of $\mathcal{X} = \{\mathcal{X}_1, ..., \mathcal{X}_5\}$.

For such generated random connection problems, we apply a two-stage method similarly as suggested in Section 6.2. That is, we first run our algorithm with the fitness function defined as (7.3) to minimize the link cost without restricting network coding. Then, from the best coding solution found, we take only the links that are used for transmission to form the subgraph to be used in the second stage. In the second stage, we use our algorithm to minimize the number of coding nodes by changing the fitness function to reflect the number of nodes where network coding is performed.

Table 7.2 shows the distribution of the found minimum number of coding nodes through the two-stage method. Again, as in the multicast case (Section 6.1.3), in most cases (about 80% of the random topologies tested) network coding is not needed at all. This time, however, the distribution of the number of required coding nodes is more spread out toward larger numbers than the multicast case (refer to Table 6.1 for comparison). This may be because now we constrained the information flow onto the calculated minimum cost subgraph, whereas in Section 6.1.3 the cost of link usage was not taken into account at all.

Table 7.2: Distribution of the calculated minimum number of coding nodes in 100 random topologies.

Number of Coding Nodes	0	1	3	4	5	6	7	9	10	13	15	17
Number of Topologies	79	6	1	1	1	3	2	2	2	1	1	1

7.5.3 Discussion

Above simulations exhibit that our evolutionary approach offers a coding strategy in the general connection problem which is still somewhat restricted but with enhanced features in many aspects, compared with existing pairwise XOR coding. For the problem of multiple unicast connections, we showed that our evolutionary approach approach yields a network coding solution that offers an advantage over traditional routing in terms of link cost, whereas an existing approach for pairwise XOR coding

may fail to provide a scalability within practical ranges as the number of connections increases. Also, we demonstrated that our evolutionary approach can tackle more general problems in which there is no restriction on the type of connection requests, taking the coding cost into account as well. Note that, though only the two-stage method was experimented for an illustrative purpose in the second set of simulations, a multi-objective evolutionary approach can also be utilized, similarly as described in Chapter 5, to investigate the tradeoff between the two objectives. In addition to the spatially distributed structure presented in Section 7.4.2, a temporally distributed structure can also be adopted for more efficient utilization of computational resources as well as more robust operation against packet losses.

A possible drawback of our evolutionary approach is that, as opposed to the case to the coding resource optimization for multicast, it lacks a performance bound. However, given that there are no practical alternatives that take into account coding among more than two flows, our approach may serve as a unique means for exploring network coding advantages in a much more generalized setup than the pairwise coding. Another possible limitation is regarding the scalability; i.e., as is typical for a GA, the population size may need to be increased significantly for large problems. However, with the distributed structure described in Section 7.4, the population size can be increased rather flexibly by increasing the size of the packet used for fitness evaluation, given that the computational complexity and memory requirement at each node scale linearly with the population size. Moreover, with the temporally distributed structure, we may further increase the effective population size by increasing k, the number of (sub)populations.

Chapter 8

Conclusion

Network coding offers numerous advantages over traditional routing while, at the same time, requiring a non-negligible amount of overhead. We believe that the crucial step toward wider deployment of network coding in practice is to show that the amount of overhead can be kept minimal and eventually be outweighed by the benefits network coding provides. Starting from the simple examples where the network coding advantage can be achieved with coding only at a subset of nodes, this thesis explores the question of when and where to code.

We first considered the problem of minimizing network coding resources for multicast, which is NP-hard. We developed an evolutionary approach based on simple GA toward a practical multicast protocol that achieves the full benefit of network coding in terms of throughput, while performing coding operations only when required at as few nodes as possible. We showed that our approach leads to a much more practical solution in the sense that it is distributed over the network both spatially and temporally, yielding a sufficiently good solution, which is at least as good as those obtained by existing approaches but often turns out to be much superior.

We then investigated the issue of tradeoff between the costs of network coding and link usage. With a slight generalization to utilize a multi-objective GA framework, we developed a method that can effectively reveal the possible tradeoff between the two costs, leading to more informed decisions on whether and where to deploy network coding.

We presented two application scenarios where our evolutionary approach can provide a number of interesting insights. First, we considered heterogeneous wireless networks where coding and non-coding legacy nodes coexist. Through simulations, we demonstrated that there may often be only a very small number of coding nodes required and suggested an effective strategy for the placement of those coding nodes. In addition, we considered the problem of fault-tolerant multicast in optical networks where the savings due to network coding may be overshadowed by the high cost of data processing in the electronic domain. Our simulations suggest that though network coding needs to be assumed to calculate a subgraph for the minimum-cost fault-tolerant multicast, the calculated minimum cost can often be achieved either without network coding at all or only at a small number of nodes; in the latter case our multi-objective evolutionary approach can lead to much more informed decisions on whether or where to employ network coding.

Interestingly, these application studies suggest that the benefit of network coding may lie in the fact that the assumption of coding can make the problem under consideration (e.g., calculation of maximal throughput, minimum cost, etc.) more tractable, but network coding may not necessarily be required to achieve the calculated advantage. This corroborates our initial intuition that the network coding advantage can often be achieved with only a very little amount of overhead, which can be considered an important merit of network coding.

We further generalized our evolutionary approach to develop a novel network coding strategy for the general connection problem beyond multicast, for which no optimal network coding strategy is known. We presented a coding strategy that allows fairly general random linear coding over a large finite field, in which decoding is done only at the receivers and the mixture of information at interior nodes is controlled by evolutionary mechanisms. We demonstrated the advantage of our coding strategy over existing end-to-end XOR coding schemes in terms of effectiveness and practicality.

There are many interesting topics for further work along the directions pursued in this thesis. An immediate extension may be to make the proposed evolutionary approaches into actual protocols, reflecting more realistic situations that may happen

in real networks. To this end, we may need to develop a mechanism that deals with exceptions such as the loss or delay of packets due to errors, collisions, topological changes, etc. Also, we may need to implement a more flexible chromosome representation at interior nodes because in real networks, it may be hard to measure the capacities of links, which may also vary over time.

There have been many recent developments in the field of evolutionary computation that significantly improve the scalability of the traditional simple GA, on which our current approaches are based. More advanced GA frameworks can be employed to find and exploit further linkage information, especially for the general connection problem considered in Chapter 7, in which, given its hardness, utilizing better linkage information beyond the coding vector level modularity may lead to far superior performance.

Another area of further investigation is the optimization of network coding resources with other constraints than throughput. Though the achievability of the maximal throughput is undoubtedly an important benefit of network coding, other characteristics such as information security or error correction capability may be of more interest in some practical networking scenarios. In such cases, we need to first set up proper fitness functions, which may involve more empirical measurements rather than simple algebraic characterizations considered in this thesis. Though we expect that the coding vector level modularity will still play an important role even with different fitness functions, a new set of genetic operators reflecting different modularity structures may need to be sought for even better performance.

Also, we may utilize the core structures of the proposed evolutionary approaches within other network problems that involve a difficult combinatorial optimization part. A key benefit of our evolutionary framework is that it can be applied even without completely characterizing the structure of the search space; rather, it operates directly on the solution itself (e.g., the actual network code in our case). However, as demonstrated in this thesis, utilizing even very limited structural properties (e.g., the first-order modularity among variables) may lead to a significant improvement in the performance of the algorithm. Along this direction, we may apply an evolutionary

approach similar to that proposed in this thesis to the scheduling problem in wireless networks with network coding. Within an evolutionary framework, we may develop a better scheduling scheme, for instance, by utilizing collisions at intermediate nodes, rather than precluding them beforehand as in most existing approaches, as they may eventually lead to a better throughput at nodes farther away. Also, we may combine evolutionary algorithms with other more traditional optimization methods to address the scaling issue in network optimization problems. For instance, for a problem in which the number of variables/constraints scales exponentially with the size of the network, an evolutionary algorithm may effectively be utilized to yield a more compact set of variables/constraints that needs to be considered at a time. Then, with traditional optimization methods being used for fitness evaluation, an optimal set of variables/constraints can be searched via evolutionary mechanisms.

Bibliography

[1] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung. Network information flow. *IEEE Trans. Inf. Theory*, 46(4):1204–1216, 2000.

[2] Boo Barkee, Deh Cac Can, Julia Ecks, Theo Moriarty, and R. F. Ree. Why you cannot even hope to use Gröbner bases in public-key cryptography? An open letter to a scientist who failed and a challenge to those who have not yet failed. *J. Symb. Comput.*, 18(6):497–501, 1994.

[3] Kapil Bhattad, Niranjan Ratnakar, Ralf Koetter, and Krishna. R. Narayanan. Minimal network coding for multicast. In *Proc. IEEE ISIT 2005.*

[4] E. Cantú-Paz and David E. Goldberg. Efficient parallel genetic algorithms: Theory and practice. *Comput. Methods Appl. Mech. Engrg.*, 186:211–238, 2000.

[5] Erick Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Parallèles, Réseaux et Systèms Répartis*, 10(2):141–171, 1998.

[6] Terence Chan and Alex Grant. Dualities between entropy functions and network codes. *submitted to IEEE Trans. Inform. Theory (arXiv:0708.4328v1)*, 2007.

[7] Zhi-Zhong Chen and Ming-Yang Kao. Reducing randomness via irrational numbers. *SIAM J. Comput.*, 29(4):1247–1256, 2000.

[8] Philip A. Chou, Yunnan Wu, and Kamal Jain. Practical network coding. In *Proc. Annual Allerton Conference on Communication, Control, and Computing*, 2003.

[9] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.*, 6(2):182–197, 2002.

[10] James. W. Demmel. *Applied Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, 1997.

[11] Berna Dengiza, Fulya Altiparmak, and Alice E. Smith. Efficient optimization of all-terminal reliable networks, using an evolutionary approach. *IEEE Trans. Rel.*, 46(1):18–26, 1997.

[12] Randall Dougherty, Christopher Freiling, and Kenneth Zeger. Insufficiency of linear coding in network information flow. *IEEE Trans. Inf. Theory*, 51(8):2745–2759, 2005.

[13] Randall Dougherty, Christopher Freiling, and Kenneth Zeger. Networks, matroids, and non-shannon information inequalities. *IEEE Trans. Inf. Theory*, 53(6):1949–1969, 2007.

[14] Randall Dougherty, Christopher Freiling, and Kenneth Zeger. Linear network codes and systems of polynomial equations. *IEEE Trans. Inf. Theory*, 54(5):2303–2316, 2008.

[15] Michelle Effros, Tracey Ho, and Sukwon Kim. A tiling approach to network code design for wireless networks. In *Proc. Information Theory Workshop*, 2006.

[16] Reuven Elbaum and Moshe Sidi. Topological design of local-area networks using genetic algorithms. *IEEE/ACM Trans. Netw.*, 4(5):766–778, 1996.

[17] Atilla Eryilmaz and Desmond S. Lun. Control for inter-session network coding. Technical Report P-2722, MIT-LIDS, 2006.

[18] Christina Fragouli and Emina Soljanin. Information flow decomposition for network coding. *IEEE Trans. Inf. Theory*, 52(3):829–848, 2006.

[19] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1993.

[20] John J. Grefenstette. Parallel adaptive algorithms for function optimization. Technical Report CS-81-19, Vanderbilt Univ. Computer Science Dept., 1981.

[21] Wayne D. Grover. Case studies of survivable ring, mesh and mesh-arc hybrid networks. In *Proc. IEEE GLOBECOM*, 1992.

[22] Georges Harik, Erick Cantú-Paz, David E. Goldberg, and Brad L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253, 1999.

[23] Tracey Ho, Yu-Han Chang, and Keesook J. Han. On constructive network coding for multiple unicasts. In *Proc. Allerton Conference on Communication, Control and Computing*, 2006.

[24] Tracey Ho, Ben Leong, Ralf Koetter, and Muriel Médard. Distributed asynchronous algorithms for multicast network coding. In *Proc. NetCod*, 2005.

[25] Tracey Ho, Ben Leong, Ralf Koetter, Muriel Médard, Michelle Effros, and David R. Karger. Byzantine modification detection in multicast networks using randomized network coding. In *Proc. IEEE ISIT*, 2004.

[26] Tracey Ho, Muriel Médard, and Ralf Koetter. An information-theoretic view of network management. *IEEE Trans. Inf. Theory*, 51(4):1295–1312, 2005.

[27] Tracey Ho, Muriel Médard, Ralf Koetter, David R. Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *IEEE Trans. Inf. Theory*, 52(10):4413–4430, 2006.

[28] Sidarth Jaggi, Michael Langberg, Sachin Katti, Tracey Ho, Dina Katabi, and Muriel Médard. Resilient network coding in the presence of byzantine adversaries. In *Proc. IEEE Infocom*, 2007.

[29] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *Proc. ACM STOC '03*, pages 355–364.

[30] Ahmed E. Kamal. 1+N protection in optical mesh networks using network coding on p-cycles. In *Proc. IEEE Globecom*, 2006.

[31] Sachin Katti, Dina Katabi, Wenjun Hu, Hariharan Rahul, and Muriel Médard. The importance of being opportunistic: Practical network coding for wireless environments. In *Proc. Allerton Conference*, 2005.

[32] Abdallah Khreishah, Chih-Chun Wang, and Ness B. Shroff. An optimization based rate control for communication networks with inter-session network coding. In *Proc. IEEE Infocom*, 2008.

[33] Ralf Koetter and Muriel Médard. An algebraic approach to network coding. *IEEE/ACM Trans. Netw.*, 11(5):782–795, 2003.

[34] Michael Langberg, Alexander Sprintson, and Jehoshua Bruck. The encoding complexity of network coding. *IEEE Trans. Inf. Theory*, 52(6):2386–2397, 2006.

[35] Daniel Lewin and Salil Vadhan. Checking polynomial identities over any field: towards a derandomization? In *Proc. ACM STOC '98*, pages 438–447.

[36] Shuo-Yen Robert Li, Raymond W. Yeung, and Ning Cai. Linear network coding. *IEEE Trans. Inf. Theory*, 49(2):371–381, 2003.

[37] Zongpeng Li and Baochun Li. Network coding: The case of multiple unicast sessions. In *Proc. Annual Allerton Conference on Communication, Control, and Computing*, 2004.

[38] Zongpeng Li, Baochun Li, Dan Jiang, and Lap Chi Lau. On achieving optimal throughput with network coding. In *Proc. IEEE Infocom*, 2005.

[39] Richard Lipton and Nisheeth Vishnoi. Deterministic identity testing for multivariate polynomials. In *Proc. SODA '03*, pages 756–760.

[40] Steven S. Lumetta and Muriel Médard. Towards a deeper understanding of link restoration algorithms for mesh networks. In *Proc. IEEE INFOCOM*, 2001.

[41] Desmond S. Lun, Muriel Médard, Tracey Ho, and Ralf Koetter. Network coding with a cost criterion. Technical Report P-2584, MIT-LIDS, 2004.

[42] Desmond S. Lun, Niranjan Ratnakar, Ralf Koetter, Muriel Médard, Ebad Ahmed, and Hyunjoo Lee. Achieving minimum-cost multicast: a decentralized approach based on network coding. In *Proc. IEEE Infocom*, 2005.

[43] Muriel Médard, Richard A. Barry, Steven G. Finn, Wenbo He, and Steven S. Lumetta. Generalized loop-back recovery in optical mesh networks. *IEEE/ACM Trans. Netw.*, 10(1):153–164, 2002.

[44] Guy Melançon and Fabrice Philippe. Generating connected acyclic digraphs uniformly at random. *Inf. Process. Lett.*, 90(4):209–213, 2004.

[45] Ronald C. Menendez and Joel W. Gannett. Efficient, fault-tolerant all-optical multicast networks via network coding. In *Proc. OFC/NFOEC*, 2008.

[46] Martin Milanič. *Report on DIMACS working group on data de-identification, combinatorial optimization, graph theory, and the stat/OR interface.* 2005.

[47] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1996.

[48] Joon-Sang Park, Mario Gerla, Desmond S. Lun, Yunjung Yi, and Muriel Médard. Codecast: A network-coding based ad hoc multicast protocol. *IEEE Commun. Mag.*, 13(5):76–81, 2006.

[49] Martin Pelikan. *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms.* Springer, 2005.

[50] S. Ramanathan. Multicast tree generation in networks with asymmetric links. *IEEE/ACM Trans. Netw.*, 4(4):558–568, 1996.

[51] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[52] Arun K. Somani. *Survivability and Traffic Grooming in WDM Optical Networks.* Cambridge University Press, 2006.

[53] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring ISP topologies with rocketfuel. In *Proc. ACM/SIGCOMM*, pages 133–145, 2002.

[54] Jay Kumar Sundararajan, Muriel Médard, Ralf Koetter, and Elona Erez. A systematic approach to network coding problems using conflict graphs. In *Proc. Information Theory and its Applications*, 2006.

[55] Danail Traskov, Niranjan Ratnakar, Desmond S. Lun, Ralf Koetter, and Muriel Médard. Network coding for multiple unicasts: An approach based on linear optimization. In *Proc. ISIT*, pages 1758–1762, 2006.

[56] Chih-Chun Wang and Ness B. Shroff. Beyond the butterfly - graph-theoretic characterization of the feasibility of network coding with two simple unicast sessions. In *Proc. IEEE ISIT*, 2007.

[57] Tsong-Ho Wu. *Fiber Network Service Survivability.* Artech House, 1992.

[58] Yunnan Wu, Philip A. Chou, and Sun-Yuan Kung. Minimum-energy multicast in mobile ad hoc networks using network coding. *IEEE Trans. Commun.*, 53(11):1906–1918, 2005.

[59] Xijin Yan, Raymond Yeung, and Zhen Zhang. The capacity region for multisource multi-sink network coding. In *Proc. IEEE ISIT*, 2007.

[60] Raymond W. Yeung. *A First Course in Information Theory,*. Kluwer Academic/Plenum Publishers, 2002.

[61] Raymond W. Yeung, Shuo-Yen Robert Li, Ning Cai, and Zhen Zhang. Network coding theory part II: Multiple source. *Foundation and Trends in Communications and Information Theory*, 2(5):330–381, 2005.