

# Evolutionary Approaches To Minimizing Network Coding Resources

Minkyu Kim, Muriel Médard, Varun Aggarwal, Una-May O'Reilly, Wonsik Kim, Chang Wook Ahn, and Michelle Effros

**Abstract**—We consider the problem of minimizing the resources used for network coding while achieving the desired throughput in a multicast scenario. Since this problem is NP-hard, we seek a method for quickly finding sufficiently good solutions. To this end, we take evolutionary approaches based on a Genetic Algorithm. In this paper, we extend the evolutionary algorithm that we previously proposed in three perspectives. First, whereas the previous algorithm can be applied to only acyclic networks, we devise a modified evaluation method that works also with networks with cycles. Second, we introduce a new set of GA components that in our experiments outperforms the one used in the previous algorithm. Third, we present a new framework of the evolutionary approach, where fitness evaluation and population management are done in a decentralized manner with a limited amount of coordination. The new framework enables a network coding protocol where the resources used for coding are optimized in the setup phase as the proposed evolutionary algorithm being loaded and run at each node of the network. We demonstrate the effectiveness of our algorithms by carrying out simulations on a number of different sets of network topologies.

## I. INTRODUCTION

It is now well known that network throughput can be significantly increased by employing the novel technique of network coding, where the intermediate nodes are allowed to combine data received from different links [1], [2]. In most of the network coding literature, it is assumed that network coding is performed at all possible nodes. However, it is often the case that network coding is required only at a subset of nodes to achieve the desired throughput.

*Example 1:* In the canonical example of network  $B$  (Fig. 1(a)) [1], only node  $z$  needs to combine its two inputs while all other nodes perform routing only. If we suppose that link  $l$  in network  $B$  has capacity 2, which we represent by two parallel unit-capacity links in network  $B'$  (Fig. 1(b)), a multicast of rate 2 is possible without network coding. In network  $C$  (Fig. 1(c)), where node  $s$  is to transmit data at rate 2 to the 3 leaf nodes, network coding is required either at node  $a$  or at node  $b$ , but not at both.  $\square$

M. Kim, M. Médard, and W. Kim are with the Laboratory of Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139, USA ({minkyu, medard, wskim14}@mit.edu).

V. Aggarwal and U.-M. O'Reilly are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA ({varun\_ag@, unamay@csail}.mit.edu).

C. W. Ahn is with Samsung Advanced Institute of Technology, Yongin, Gyeonggi 446-712, Korea (cwan.ahn@samsung.com).

M. Effros is with the Data Compression Laboratory, California Institute of Technology, Pasadena, CA 91125, USA (effros@caltech.edu).

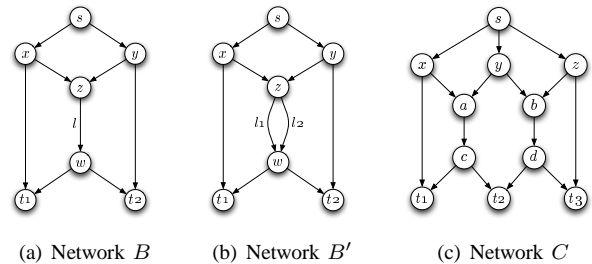


Fig. 1. Sample Networks for Example 1

The above example leads us to the following question: To achieve the desired throughput, at which nodes does network coding need to occur?

If network coding is handled at the application layer, we can minimize the performance penalty incurred by network coding by identifying the nodes where access up to the application layer is not necessary. On the other hand, if network coding is performed as integrated in the buffer management of routers, it is of natural interest to reduce the number of such special devices deployed while satisfying the communication demand.

The problem of determining a minimal set of nodes where coding is required is NP-hard: its decision problem, which decides whether the given multicast rate is achievable without coding, reduces to a multiple Steiner subgraph problem, which is NP-hard [3]. It is shown that even approximating the minimum number of coding nodes within any multiplicative factor or within an additive factor of  $|V|^{1-\epsilon}$  is NP-hard [4].

As in the above example of network  $C$ , whether coding is necessary at a node should be considered jointly with other nodes. Thus, the problem of deciding where to perform network coding in general involves a selection out of exponentially many possible choices.

For this problem, Kim *et al.* [5] propose an evolutionary approach, based on a Genetic Algorithm (GA), that manages a set of candidate solutions of a suitably small size, sequentially enhancing these solutions in an evolutionary manner. Reference [5] demonstrates the benefit of the evolutionary approach over other existing approaches in terms of the performance in reducing the number of coding links/nodes and its applicability to a variety of generalized scenarios.

In this paper, we extend the results of [5] in three perspectives. First, whereas the algorithm in [5] can be applied to only acyclic networks, we devise a modified evaluation method that works also with networks with cycles. Second, we introduce

a new set of GA components that in our experiments outperforms the one used in [5]. Third, we present a new framework of the evolutionary approach, where fitness evaluation and population management are done in a decentralized manner. The new framework enables a network coding protocol where the resources used for coding are optimized in the setup phase as the proposed evolutionary algorithm being loaded and run at each node of the network.

The rest of the paper is organized as follows. Section II presents the problem formulation with related work and Section III describes the centralized approach, which Section IV extends to a semi-decentralized version. Section V gives the results of the numerical experiments and Section VI concludes with a summary of the results.

## II. PROBLEM FORMULATION AND RELATED WORK

### A. Problem Formulation

We assume that a network is given by a 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. We consider the single multicast scenario in which a single source  $s \in V$  wishes to transmit data at rate  $R$  to a set  $T \subset V$  of sink nodes, where  $|T| = d$ . Rate  $R$  is said to be achievable if there exists a transmission scheme that enables all  $d$  sinks to receive all of the information sent. We only consider linear coding, where a node's output on an outgoing link is a linear combination of the inputs from its incoming links. Linear coding is sufficient for multicast [2].

Given an achievable rate  $R$ , we wish to determine a minimal set of nodes where coding is required in order to achieve this rate. However, whether coding is necessary at a node is determined by whether coding is necessary at at least one of the node's outgoing links. Hence, as pointed out also in [4], the number of coding links is, in fact, a more accurate estimator of the amount of computation incurred by coding.

We assume hereafter that our objective is to minimize the number of coding *links* rather than *nodes*. Note, however, that as demonstrated in [5], it is straightforward to generalize the proposed approaches to the case of minimizing the number of coding nodes. Furthermore, [5] shows that, with appropriate changes, the approaches can be readily applied to more generalized optimization scenarios.

It is clear that no coding is required at a node with only a single input since these nodes have nothing to combine with (a formal proof appears in [6]). For a node with multiple incoming links, if the linearly coded output to a particular outgoing link weights all but one incoming message by zero, effectively no coding occurs on that link; even if the only nonzero coefficient is not identity, there is another coding scheme that replaces the coefficient by identity [4].

Let us refer to a node with multiple incoming links as *merging node*. To determine whether coding is necessary at an outgoing link of a merging node, we need to verify whether we can restrict the output of the link to depend on a single input without destroying the achievability of the given rate.

At each merging node, for this verification at each of its outgoing link, we assign a binary variable to each incoming link, where 1 indicates that the input from the associated link contributes to the output, and 0 indicates that it does not. Hence, if we denote by  $d_{in}^v$  and  $d_{out}^v$  the in-degree and the out-degree of node  $v$ , we have a total of  $m = \sum_{v \in V} d_{in}^v d_{out}^v$  binary variables and we are to explore the  $m$ -dimensional binary space to find the desired minimal set of coding nodes.

To this end, we employ a GA-based search method that serves to reduce efficiently the size of the space to be actually searched using an evolutionary mechanism.

### B. Related Work

Fragouli *et al.* [7] show that coding is required at no more than  $d - 1$  nodes in acyclic networks with 2 unit-rate sources and  $d$  sinks. 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  $d$  sinks. They then construct the corresponding labeled line graph and sequentially remove the links whose removal does not affect the achievable rate.

Langberg *et al.* [4] 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.

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. Note that, unless a good order of the link traversal is found by some other method, the quality of the solution cannot be much improved even at the expense of additional computational resources.

Bhattach *et al.* [8] 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 sinks, which makes it hard to apply the formulations to the case of a large number of sinks, even at the price of sacrificed optimality.

We conclude this section with a brief introduction to GA.

### C. A Brief Introduction to GA

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., [9], [10]).

GAs [11], [12] operate 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 fitter chromosomes are selected with higher probability. For crossover, survived 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 a GA. The above process is iterated with the newly generated population successively replacing the current one. The GA terminates when a certain stopping criterion is reached, e.g., after a predefined number of generations.

There are several aspects of our problem suggesting that a GA-based method may be a promising candidate: GA has proven to work well if the space to be searched is large, but known not to be perfectly smooth or unimodal, or if the space is not well understood, and if finding a global optimum is not critical [11]. Note that the search space consisting of  $m$ -dimensional binary vectors is not smooth or unimodal with respect to the number of coding links and the structure of the space consisting of the feasible vectors is not well understood. Given the NP-hardness of the problem, it is not critical that the calculated solution may not be optimal.

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 nodes therein efficiently 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.

### III. CENTRALIZED APPROACH

We first present the centralized version of the evolutionary approach which is applicable when the size of the network is moderate and the topology of the whole network is known to the central location of computation. Also, the centralized algorithm is appropriate if a possibly long time of computations is acceptable, e.g., in the planning stage of the network.

Fig. 2 shows the overall structure of our centralized algorithm, which is based on the simple GA [11]. In the following, subsections III-A and III-B present two different methods for representing the object of optimization as a chromosome (item C1 in Fig. 2) and evaluating the chromosomes (items C3 in Fig. 2). Subsection III-C then describes the computational part of the algorithm (remaining items in Fig. 2), which can be combined with either of the two methods in the preceding subsections.

#### A. Algebraic Method

This is the representation and evaluation method presented in [5]. Despite the limitation that the method can be applied only when the given network is acyclic, here we briefly present

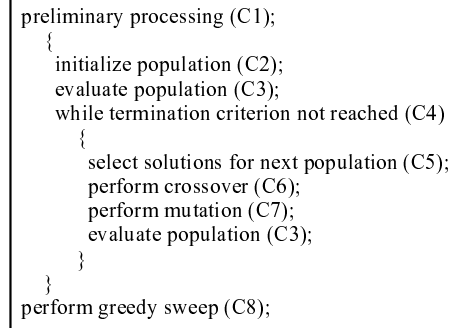


Fig. 2. Structure of the Centralized Algorithm

its main concepts; the reader is referred to [5] for details. Note that, as can be seen in the next section, the main ideas in this subsection will be used in the semi-decentralized approach.

We first construct the labeled line graph  $G' = (V', E')$  corresponding to  $G$  [13]. In  $G'$ , we refer to each node with multiple incoming links as a *coding point* and let  $C$  be the set of all coding points. For the  $i$ th coding point  $c_i \in C$ , we let  $\chi_i$  be the set of coefficients associated with the incoming links to  $c_i$ , and let  $\chi$  denote the union of all such  $\chi_i$ 's where  $|\chi| = m$ . We assume that the components of vector  $\underline{\xi}$ , the set of all link coefficients, are rearranged such that the first  $m$  components of  $\underline{\xi}$  belong to  $\chi$ , i.e.,  $\xi_j \in \chi (1 \leq j \leq m)$ .

Each chromosome is an  $m$ -dimensional binary vector, whose  $k$ -th component is associated with  $\xi_k$  ( $1 \leq k \leq m$ ). Once a chromosome  $\underline{y}$  is given, we refer to each coding point  $c_i$  as *inactive* if the number of 1's in the  $\underline{y}$ 's components associated with the set  $\chi_i$  is at most one (i.e., coding is not happening), and *active* otherwise.

For a given chromosome  $\underline{y}$ , we verify its feasibility by evaluating the system matrix for each of the  $d$  sinks while the components of  $\underline{\xi}$  associated with 0's in  $\underline{y}$  being zeroed out [5]. This feasibility test entails a bounded error, which is shown in [5] not critical since the error is one-sided, i.e., a feasible chromosome may mistakenly be declared infeasible but not in the opposite way, and also the probability bound of the error can be lowered as much as we desire at an additional cost of computation. We then define the fitness value  $F$  of chromosome  $\underline{y}$  as

$$F(\underline{y}) = \begin{cases} \text{number of active } c_i\text{'s,} & \text{if } \underline{y} \text{ is feasible,} \\ \infty, & \text{if } \underline{y} \text{ is infeasible.} \end{cases} \quad (1)$$

For the evaluation of a single chromosome, computational complexity of  $O(d \cdot (\mathcal{M}(|E|) + R^3))$  is required, where  $|E|$  is the number of links in the network and  $\mathcal{M}(|E|)$  is the complexity required to multiply two  $|E|$  by  $|E|$  matrices, whose best bound so far is  $\mathcal{M}(|E|) = O(|E|^{2.376})$  [14].

#### B. Graph Decomposition Method

Note that the above algebraic method deals explicitly 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, and the system matrix, defined as  $M = A(D)(I - DF)^{-1}B(D)$ , becomes a matrix over the polynomial ring with coefficients being *rational functions* in the delay variable  $D$  [13].

We can determine the achievability of the multicast rate by examining whether the determinant of  $M$  for each connection is nonzero over the *field of rational functions*. Note, however, that the elements of the matrices  $A(D)$ ,  $B(D)$ , and  $(I - DF)^{-1}$  are rational functions in  $D$ . Therefore, the matrix computations are to be done by calculating the coefficient for each power of the delay variable  $D$ , which in general is much more complex and requires more resources than the simple algebraic computations in the delay-free case. Hence, direct manipulation of the transfer matrices for the feasibility tests as in the above algebraic method may become prohibitively inefficient for networks with cycles.

In this subsection, we show that, with an appropriate graph decomposition, the evaluation can be done by calculating the max-flows between the source and each of the sinks. Note that the claim that the minimum of those max-flows equals the maximum achievable multicast rate holds regardless whether the network is acyclic or not [1].

We first note that, unlike the algebraic approach, this modified method operates on the actual graph of the given network rather than on the corresponding labeled line graph. In the first stage of the algorithm (item C1 in Fig. 2), we decompose all merging nodes that are not a sink as follows.

Consider a merging node  $v$  with  $d_{in} (\geq 2)$  incoming links and  $d_{out}$  outgoing links (see Fig. 3). We introduce  $d_{in}$  nodes  $u_1, \dots, u_{d_{in}}$ , which we call *incoming auxiliary nodes*, and redirect the  $i$ -th ( $1 \leq i \leq d_{in}$ ) incoming link of node  $v$  to node  $u_i$ . Similarly, we create  $d_{out}$  nodes  $w_1, \dots, w_{d_{out}}$ , which we call *outgoing auxiliary nodes*, and let the  $j$ -th ( $1 \leq j \leq d_{out}$ ) outgoing link of node  $v$  be the only outgoing link of node  $w_j$ . We then insert a link  $(u_i, w_j)$  between each pair of nodes  $u_i$  and  $w_j$  ( $1 \leq i \leq d_{in}$ ,  $1 \leq j \leq d_{out}$ ). Hence, for each merging node with in-degree  $d_{in}$  and out-degree  $d_{out}$ , the number of nodes increases by  $(d_{in} + d_{out} - 1)$  and the number of links by  $d_{in} \cdot d_{out}$  after the decomposition.

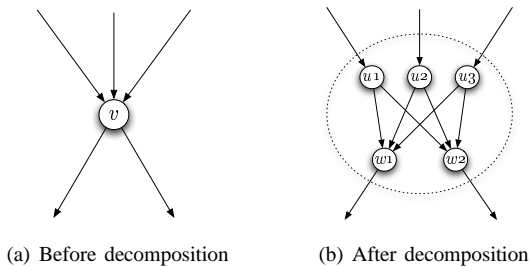


Fig. 3. Decomposition of a node with  $d_{in} = 3$  and  $d_{out} = 2$ .

With each of the newly introduced links between the auxiliary nodes, we associate a binary variable in the chromosome. Recall that in the algebraic method, a binary variable is assigned to each coefficient associated with an incoming link

of a merging node in the labeled line graph. Since a link in the labeled line graph corresponds, in the original graph, to a pair of incoming and outgoing links connected through a node, we can see a one-to-one correspondence between the binary variables in the chromosomes of the two methods.

To verify the feasibility of a given chromosome  $\underline{y}$ , we first delete all the links associated with 0's in  $\underline{y}$ . If an outgoing auxiliary node has at most one incoming link, we refer to the outgoing auxiliary node as *inactive*, and *active* otherwise. Note that we can replace each inactive outgoing auxiliary node, together with its only incoming link and outgoing link, by a single link without changing any of the max-flows; if it has no incoming link, we simply delete the node. Hence, the number of inactive outgoing auxiliary nodes corresponds to the number of non-coding links.

We then compute the max-flow between the source and each of the  $d$  sinks. If all  $d$  max-flows are at least  $R$ , the chromosome is declared feasible, and otherwise infeasible. Since we are to minimize the number of coding links, we define the fitness value  $F$  of the given chromosome  $\underline{y}$  as

$$F(\underline{y}) = \begin{cases} \text{number of active} \\ \text{outgoing auxiliary nodes,} & \text{if } \underline{y} \text{ is feasible,} \\ \infty, & \text{if } \underline{y} \text{ is infeasible.} \end{cases}$$

Note that, unlike the algebraic method, this evaluation method does not use randomly generated codes for feasibility test and thus it incurs no error. However, code construction needs to be done separately after determining the coding links.

The max-flow-based evaluation of a single chromosome requires  $O(d \cdot |E'|^2 \sqrt{|V'|})$  [15], where  $|E'|$  and  $|V'|$  are the number of links and nodes, respectively, in the decomposed network. Note that  $|E'|$  and  $|V'|$  can be much larger than the corresponding parameters in the original network if the network is highly connected.

### C. Computational Part of GA

1) *Block-Wise Operations*: In [5], a chromosome of length  $m$  is allowed to take any of  $2^m$  possible strings of length  $m$ . Let us consider a particular subset of the chromosome consisting of  $k$  bits that correspond to an outgoing link fed by  $k$  incoming links, which we refer to as *block* of length  $k$ . Note that whether coding is done for the outgoing link depends on whether the number of 1's in the block is at least two or not. Once the block has at least two 1's, replacing all the remaining 0's with 1's has no effect on whether coding is done. Moreover, substituting 0 with 1, as opposed to substituting 1 with 0, does not hurt the feasibility. Therefore, in terms of our objective of the optimization, any block with two or more 1's can be treated the same as the block with all 1's.

Let us now introduce a new structure for representing chromosomes, which we refer to as *block structure*, where each block of length  $k$  can only take one of the following  $(k + 2)$  strings:  $[000\dots 0]$ ,  $[100\dots 0]$ ,  $[010\dots 0]$ ,  $[001\dots 0]$ , ...,  $[000\dots 1]$ ,  $[111\dots 1]$ . Note that the size of the space for a block of length  $k$ , which is exponential without the block structure, becomes linear.

When initializing the population (item C2 in Fig. 2), each block is randomly generated. Then, an all-one vector is inserted to the population, whose role as a feasible starting point is critical to the performance of the algorithm, as discussed in [5].

To preserve the block structure throughout genetic operations, we also need to devise a new set of genetic operators. In [5], parameterized uniform crossover [11] and simple binary mutation [11] are used; in parameterized uniform crossover, two chromosomes selected for crossover exchange each bit independently with a given probability, and in simple binary mutation, each bit of a chromosome is flipped independently with a given probability. Note that both of operators are applied to each *bit* of the chromosome, hence we refer to these operators as *bit-wise* operators.

We now define the *block-wise* operators. For block-wise uniform crossover, we let two chromosomes subject to crossover now exchange each block, rather than bit, independently. Block-wise mutation, however, needs a more radical change since each block has more than just binary choices: each block takes another allowed string uniformly at random.

For a length- $k$  block, if mutation rate is  $\alpha$ , the average number of flipped bits by mutation is  $k\alpha$  for bit-wise mutation, and on the other hand,  $\frac{4k^2}{(k+1)(k+2)}\alpha$  for block-wise mutation, as can be easily verified. While the difference becomes more apparent when  $k$  is large since the latter is upper bounded by  $4\alpha$ , those values are still different for  $k = 2$ , where the block structure makes no difference in the space size. Though block-wise mutation may lead to much smaller number of flipped bits, it is more likely to cause a sudden change in a chromosome, which may negatively affect GA's ability to improve the solution through fine random changes.

Hence, though the block-wise operations greatly reduce the size of the space to be searched, their overall effect on the performance of GA is not easy to predict theoretically. The effect will be evaluated later by empirical tests.

2) *Greedy Sweep*: After the iteration terminates, we further inspect the best chromosome produced to switch each of the remaining 1's to 0 if it can be done without violating feasibility (item C8 in Fig. 2). Let us refer to this procedure as *greedy sweep*. This procedure may only improve the solution, and sometimes the improvement can be substantial, as will be shown in Section V.

If we refer to the chromosome obtained after greedy sweep as  $\underline{z}$ ,  $\underline{z}$  is minimal in the sense that any chromosome formed from  $\underline{z}$  by switching 1 to 0 is infeasible. Recall that, in the evaluation method in Section III-B, a 0 in the chromosome means that the associated link in the decomposed graph will be deleted. Hence, the decomposed graph associated with  $\underline{z}$  is minimal with respect to link removal. One can easily verify that there is a one-to-one correspondence between the links between auxiliary nodes in our decomposed graph (see Fig. 3) and the paths within the gadget  $\Gamma_v$  (see Fig. 1 in [4]). Therefore, from  $\underline{z}$ , we can construct the corresponding simple instance, as defined in [4]. Note that a simple instance gives the upper bounds on the number of coding links [4] (in [4], the

number of coding links is the same as the number of coding nodes) and the bounds also apply to  $\underline{z}$ , which leads to the following lemma:

*Lemma 1*: The number of coding links associated with  $\underline{z}$  is upper bounded by 1)  $R^3 d^2$  in acyclic networks and 2)  $(2B + 1)R^3 d^2$  in cyclic networks, where  $B$  is the minimum number of links that must be removed from the network in order to eliminate cycles.

Hence, our algorithm performs no worse than the algorithm in [4], though in most cases our algorithm outperforms the one in [4] even without greedy sweep. Greedy sweep turns out to be useful when, e.g., the parameters of GA may not be adjusted properly. This phenomenon is demonstrated in Section V.

3) *Other Components of GA*: The iteration is terminated if the generation number reaches the predefined limit (item C4 in Fig. 2). For selection mechanism (item C5 in Fig. 2), we employ *tournament selection* [11]: we repeat the random match where the best chromosome out of the predefined number of randomly selected chromosomes is selected with replacement for the next generation. Our experimental tests show that tournament selection, compared with the rank-based selection used in [5], offers better solution quality, as well as faster running time due to not having to sort the chromosomes.

#### IV. SEMI-DECENTRALIZED APPROACH

The size of the population often serves as an important factor for the ability of a GA to find a good solution [16]. 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. Though it is not an easy task to predict the accurate population size required for a specific problem, it is always desirable to devise an evaluation method with a low complexity, which allows for a flexibility in adopting a large-sized population when required.

Note that for the centralized approach in the previous section, the computational complexity required for the evaluation of a chromosome depends on the size of the whole network with exponents larger than 2, i.e.,  $O(|E|^{2.376})$  or  $O(|E'|^2 \sqrt{|V'|})$ . As the size of the network grows, this complexity issue may prevent the population size from scaling appropriately.

In this section, we present a modified framework for the evolutionary approach, in which the feasibility test is done locally at each sink, while the intermediate nodes simply compute random linear combinations of inputs. With a limited amount of feedback information from the sinks and the merging nodes, the evaluation can be done with a significantly lower complexity.

Furthermore, the population can be managed in a distributed manner such that each merging node locally manages a subset of the population. Note that each binary variable in a chromosome determines the local operation at an individual node (see Fig. 4). Hence, whether coding is done can be checked locally at each merging node. Also, as the details will

be discussed later, all genetic operations can be done locally at individual merging nodes. Thus, we refer to this approach as *semi-decentralized*.

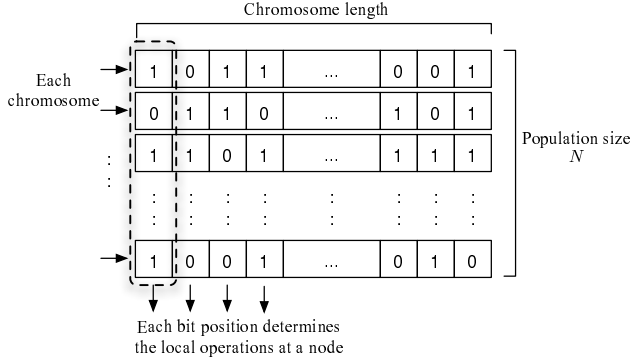


Fig. 4. Structure of the Population

Note that, in addition to the ability to deal with a large-sized population, this semi-decentralized approach has an important benefit that the minimal set of coding links/nodes can be obtained on the fly while a communication network is operational. Hence, the following optimization protocol can be readily developed. All nodes participating in the transmission are loaded with the proposed semi-decentralized algorithm. As the source node sends a "start" signal, all the nodes go into the optimization mode starting to run the algorithm. The nodes where coding is not required are identified as the algorithm progresses, and when the source node sends a "stop" signal, the network starts to transmit data with the identified coding and non-coding nodes.

We need to take different approaches depending on whether the network has cycles or not. In the following, we begin to describe the details of the semi-decentralized approach assuming that the network is acyclic. Later in this section, we extend the approach to the networks with cycles, highlighting the changes to be made.

#### A. Notations and Preliminaries

We assume that the following parameters are determined and shared before the algorithm starts: the target multicast rate  $R$ , the population size  $N$ , the field size  $q$ , the crossover probability, and the mutation rate.

Let us define several data structures to be used in the algorithm. For the feasibility test of a chromosome, each node communicates a vector consisting of  $R$  components, which we refer to as *pilot vector*. Each of the components is from  $\mathbb{F}_q$  and the  $i$ -th component represents the coefficient to multiply with the  $i$ -th source process. We assume that each node transmits a set of  $N$  pilot vectors at a time so that the feasibility test of the whole population is done all together.

Let us consider a merging node with  $d_{in} (\geq 2)$  incoming links. For *each* of its outgoing links, the node manages  $N$  binary vectors of length  $d_{in}$ , each of which we refer to as *coding vector*. The  $i$ -th ( $1 \leq i \leq d_{in}$ ) component of  $j$ -th

( $1 \leq j \leq N$ ) coding vector determines whether the input from the  $i$ -th incoming link contributes to the  $j$ -th randomly coded output; i.e., it corresponds to the link coefficient between the  $i$ -th incoming link and the  $j$ -th outgoing link.

For the feedback of the feasibility and the number of coding links, each node transmits a vector consisting of  $N$  components, which is referred to as *fitness vector*. Each of the components must be at least  $\lceil \log(|E| + 2) \rceil$  bits long; for each chromosome, the number of coding links can be from zero to  $|E|$  and an additional symbol (infinity) is needed to signify infeasibility.

Since the population is to be managed at the merging nodes, the genetic operations are also to be done at the merging nodes individually, but with some coordination. The source calculates and sends a *coordination vector* consisting of the following information: the indices of the selected chromosomes, the random pairings of the selected chromosomes, and whether or not each pair is subject to crossover.

The overall structure of our semi-decentralized algorithm is shown in Fig. 2 with the locations specified where each procedure is to be performed.

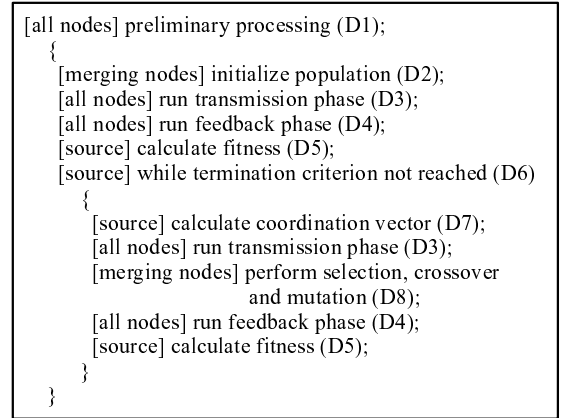


Fig. 5. Structure of the Semi-Decentralized Algorithm

Note that the semi-decentralized algorithm does not employ the greedy sweep procedure as it needs a global coordination. However, as the semi-decentralized algorithm offers significantly lower complexity than the centralized algorithms in the previous section, we have much more freedom to choose a larger-sized population to improve the solution quality.

#### B. Initialization

Each merging node with in-degree  $d_{in} (\geq 2)$  and out-degree  $d_{out}$  generates  $N \cdot d_{in} \cdot d_{out}$  coding vectors randomly (item D2 in Fig. 5). As in the centralized algorithm, an all-one vector is inserted, which is done by setting the first component of each coding vector to 1.

#### C. Fitness Evaluation

For fitness evaluation (items D3 and D4 in Fig. 5), each intermediate node operates in a burst-oriented mode; i.e., for

the transmission (feedback) phase, each node starts updating its output only after an updated input has been received from all incoming (outgoing) links.

1) *Transmission Phase*: The source node transmits on each of its outgoing links a set of  $N$  random pilot vectors. Each non-merging node simply forwards the  $N$  pilot vectors received from its incoming link to all its outgoing links.

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

$$J = \{1 \leq j \leq d_{in} \mid \text{the } i\text{-th component of } v_j \text{ is } 1\}. \quad (2)$$

Then,

$$u_i = \sum_{j \in J} w_j \cdot \text{rand}(\mathbb{F}_q), \quad (3)$$

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

2) *Feedback Phase*: In order for the source node to calculate the fitness values of a chromosome, two kinds of information need to be gathered: 1) the achievability of rate  $R$  at each of the sink nodes and 2) the number of coding links. Since this information is needed only at the end of a generation for selection, the feedback of the information is done once in a generation after the transmission of  $N$  pilot vectors.

Each sink determines whether the target rate  $R$  is achievable for each of the  $N$  chromosomes by inspecting the received pilot vectors; i.e., if the rank of the collection of the  $i$ -th ( $1 \leq i \leq N$ ) pilot vectors is  $R$ , rate  $R$  is achievable for the sink.

Regarding the number of coding links, each merging node can count its number of coding links by inspecting the coding vectors used in the transmission phase. Note that the information to be reported to the source node is the sum of those numbers, not the specific number of coding links at each merging node. This feedback is done as follows.

- i) After the feasibility tests of the  $N$  chromosomes are done, each sink generates a fitness vector where the  $i$ -th ( $1 \leq i \leq N$ ) component is zero if the  $i$ -th chromosome is feasible at the sink, and infinity otherwise. Each sink then initiates the update phase by transmitting its fitness vector to all of its parents.
- ii) Based on the received vectors, each node calculates its own fitness vector such that the  $i$ -th ( $1 \leq i \leq N$ ) component is the number of coding links of the node for the  $i$ -th chromosome plus the sum of all the  $i$ -th components of the received vectors. Each node then transmits the calculated fitness vector to *only one* of its parents, and to the other parent nodes sends an all-zero fitness vector.

- iii) The source node, after receiving an updated vector from all incident links, calculates its fitness vector by performing component-wise sum of all the received vectors.

Note that, since the network is assumed to be acyclic, each coding link of a chromosome contributes exactly once to the corresponding component of the source node's fitness vector, and thus the above update procedure provides the source with the correct total number of coding links.

3) *Fitness Calculation*: Based on the fitness vector, the source node calculates the fitness values of  $N$  chromosomes (item D5 in Fig. 5) such that the fitness value of the  $i$ -th chromosome is simply the  $i$ -th component of the fitness vector; if an infinity is generated by any of the sinks, it is delivered unchanged to the source node. Note that it is the correct fitness value for an infeasible chromosome (cf. (1)).

#### D. Genetic Operations

Once the fitness values are calculated, a coordination vector can be calculated at the source (item D7 in Fig. 5). The coordination vector, which is to be renewed for each generation as the pilot vectors, is transmitted with the pilot vectors in the transmission phase. Thus, there is no extra broadcast scheme required to share the coordination vector. Assuming that the coordination vector is received, the population for the next generation can be constructed independently at each of the merging nodes as follows (item D8 in Fig. 5):

- For selection, each merging node stores the coding vectors used for the current generation, out of which each merging node can simply pick the components that correspond to the indices of the selected chromosomes.
- In block-wise uniform crossover, whether each block is exchanged between a pair of chromosomes is determined independently with other blocks. Since no block is shared by multiple merging nodes, crossover operations can be done independently at each merging node.
- Block-wise mutation neither requires any coordination between different bit blocks, which leads to independent mutation operations at each emerging node.

#### E. Complexity

Each merging node  $v$  computes random linear combinations of inputs, which requires  $O(d_{in}^v \cdot d_{out}^v \cdot R)$ . The feasibility test at each sink  $t$  is done by calculating the rank of an  $h_t \times R$  matrix, where  $h_t$  is the number of incoming links of  $t$  and we assume  $h_t \geq R$ , hence it requires  $O(h_t^2 R)$ . In the feedback phase, the update of a fitness vector takes  $O(d_{in}^v + d_{out}^v)$  per chromosome. Therefore, the evaluation of a single chromosome requires the computational complexity of  $O(\sum_{v \in V} d_{in}^v \cdot d_{out}^v \cdot R) + O(\sum_{t \in T} h_t^2 R)$ .

#### F. Networks with Cycles

Cycles can be dealt with in two different ways as in other network coding problems. First, we can select a subgraph that does not contain a directed cycle, based on which we proceed to code construction and decoding in essentially the same

manner as in the acyclic case [17], [18]. Alternatively, we can directly apply coding over cycles by combining information from possibly different time periods at intermediate nodes and deploying memory at the receivers for decoding [13], [19], [20], where the network code can be considered essentially a convolutional code.

The former of the above two scenarios allows for simple coding and decoding, but it may necessitate coding at the links/nodes where coding is not necessary if some link connections were not removed in the earlier stage [5]. On the other hand, the latter scenario may allow us to explore the full-fledged tradeoff between coding and capacity, but both specifying and decoding the code are more complex than in the former case.

Recall, however, that the important feature of this semi-decentralized approach is that it can be carried out as a part of the setup protocol within the framework of the network coding scheme used. If the original coding scheme is designed to operate on the acyclic subgraph selected beforehand, there is no reason to employ more complex network codes based on the original graph with cycles to minimize coding resources.

In the following, we describe how our semi-decentralized approach can be incorporated in the former of the above two scenarios. Note, however, that a similar approach can be readily applied to the convolutional network coding scenario with an appropriate cycle-avoiding mechanism for the transmission of the control messages such as the feedback information.

To set up an acyclic set of connections on a given network, we use the distributed algorithm presented in [18], where a binary variable is assigned to each pair  $(l, l')$  of incident links indicating that the connection from link  $l$  to link  $l'$  is allowed or not, if the variable is 1 or 0, respectively. The value of each binary variable is determined such that the transmission along a directed cycle is prohibited.

It is interesting to note that the binary variables used for subgraph selection are actually assigned to the link coefficients as in our method. Hence, our method works in two stages as follows:

- i) Use the algorithm in [18] to select the set of link coefficients to be used for transmission. Each node then exchanges the binary variables assigned to its links with its neighbors.
- ii) We then apply the evolutionary method as described in sections from IV-A to IV-D using only those link coefficients selected in the first stage.

Alternatively, if minimizing the link cost is the primary concern, one may need to first select a subgraph that achieves the desired throughput with the minimum cost using a decentralized algorithm in [21]. Such a subgraph would not contain a directed cycle if the network is assumed to have no links with zero or negative costs. Hence, another two-stage method is possible where the minimum-cost subgraph selection is followed by the evolutionary method with the changes as above. The performance of this two-stage method is demonstrated in the next section.

## V. EXPERIMENTAL RESULTS

For the simulations in this section, the population size is set to 100 and the iteration is terminated after 1000 generations. The size of the selection tournaments is 90. The rate of crossover is fixed at 0.8 and the mutation rates that yield the best performance at empirical tests are 0.003 and 0.006 for block-wise and bit-wise mutations, respectively.

### A. Effects of Block-Wise Operations and Greedy Sweep

To test the effect of the block-wise operations introduced in Section III-C, we employ the topologies generated by the algorithm in [22], which constructs connected acyclic directed graphs uniformly at random. We artificially create bottlenecks by greedily removing some portion of redundant links, i.e., the links whose removal does not affect the multicast rate. Two networks with parameters (50 nodes, 87 links, 10 sinks, rate 5) and (75 nodes, 156 links, 15 sinks, rate 7) are used.

We use the centralized version of our algorithm with the graph decomposition method, using the block-wise operations and the bit-wise operations. To compare the performance of our algorithm with other approaches, we perform numerical tests using the two previously mentioned minimal approaches by Fragouli *et al.* [7] ("Minimal 1") and Langberg *et al.* [4] ("Minimal 2"), in both of which link removal is done in a random order. Table I shows the best and the average values obtained in 20 random trials.

	(50,87,10,5)		(75,156,15,7)	
	Best	Avg.	Best	Avg.
Block-wise	2	2.50	3	4.30
Bit-wise	2	3.30	3	5.75
Minimal 1	3	5.00	10	21.20
Minimal 2	3	4.35	9	11.40
Block-wise (w/o greedy sweep)	2	2.50	3	4.40
Bit-wise (w/o greedy sweep)	2	4.50	24	30.05

TABLE I  
COMPARISON OF NUMBERS OF REQUIRED CODING LINKS

In our experiments, our algorithm with the block-wise outperforms the one with the bit-wise operations. One can observe in the table that the performance of our algorithm, with either of the genetic operations, is everywhere at least as good and often far better than that of both Minimal 1 and Minimal 2 both in the best and in the average values.

To demonstrate the effectiveness of the greedy sweep procedure, we also show the solutions by our algorithm without greedy sweep. One can notice that, without greedy sweep, our algorithm with the bit-wise operations may perform worse than the minimal approaches. Such poor performance may be due to that some parameters, such as population size, are not suitably chosen. Even with such misadjusted parameters, our algorithm, if combined with greedy sweep, performs much better than the minimal approaches.

### B. Performance of Semi-Decentralized Approach

Since the centralized and the semi-decentralized approaches share the same computational part of GA, they show the same

average performance in terms of the quality of the solution. One can observe, however, a significant gain in terms of complexity with the semi-decentralized approach (cf. Sections III-A, III-B, and IV-E).

We generate a set of highly connected topologies such that there exists a link between each pair of nodes  $i$  and  $j$  ( $i < j$ ), where the source is node 1 and the sinks are the last 10 nodes, and compare the running times of the two approaches. This test is pessimistic in the sense that the decentralized algorithm is run on a single machine with each node's function emulated by a separate thread, thus it does not benefit from the possible multi-processing gain whereas it suffers from additional burdens incurred by the management of a large number of threads. Table II shows the elapsed time (in seconds) per generation for the two approaches. We observe that the semi-decentralized approach nevertheless offers the advantage in running time as the size of the network grows.

Number of nodes	15	20	25	30	35	40
Centralized	0.3	1.5	4.3	13.5	29.5	65.6
Semi-decentralized	1.8	2.7	4.4	6.3	10.8	15.4

TABLE II  
ELAPSED TIME PER GENERATION

### C. Effectiveness of Two-Stage Method

In Section IV-F, we introduced the two-stage method where we first select the minimum-cost subgraph assuming network coding is done everywhere and based on the resulting subgraph we apply our evolutionary algorithm to minimize the network coding resources. Though it is not optimal, this two-stage method can be very useful when optimization over both link cost and coding cost is required; the minimum link cost is guaranteed, and the resulting subgraph is acyclic and can be substantially smaller than the original network.

We use the algorithm in [21] to calculate the minimum cost subgraph of the topologies of ISP 1755 and ISP 3967 obtained from the Rocketfuel project [23]. On the resulting subgraphs, we apply our centralized algorithm with block-wise operations to minimize the number of coding links.

For ISP 1755 with target rates 2, 3, and 4, each run always yields *zero* coding links required, though there are numerous merging nodes, i.e., the possible places for coding, in the resulting subgraph after the first stage. We have the same result for ISP 3967 with rates 2 and 3.

These results suggest that the two-stage method may be, not only computationally efficient, very effective in practice to find an efficient multicast scheme with the both criteria of link cost and coding cost.

## VI. CONCLUSIONS

We presented the evolutionary approaches to minimizing network coding resources that extend the results in [5] in three perspectives. First, we devised a modified evaluation method that works also with networks with cycles. Second, we introduced a new set of GA components that outperforms

the one used in [5]. Third, we presented a semi-decentralized framework of the evolutionary approach that enables a network coding protocol, where the resources used for coding are optimized in the setup phase as the proposed evolutionary algorithm being loaded and run at each node of the network. We then demonstrated the effectiveness of our algorithms by carrying out simulations on a number of different sets of network topologies.

## REFERENCES

- [1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inform. Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.
- [2] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. Inform. Theory*, vol. 49, no. 2, pp. 371–381, 2003.
- [3] M. B. Richey and R. G. Parker, "On multiple steiner subgraph problems," *Networks*, vol. 16, no. 4, pp. 423–438, 1986.
- [4] M. Langberg, A. Sprintson, and J. Bruck, "The encoding complexity of network coding," in *Proc. IEEE ISIT '05*.
- [5] M. Kim, C. W. Ahn, M. Médard, and M. Effros, "On minimizing network coding resources: An evolutionary approach," in *Proc. NetCod*, 2006.
- [6] Y. Wu, P. A. Chou, and S. Kung, "Minimum-energy multicast in mobile ad hoc networks using network coding," *IEEE Trans. Commun.*, to appear.
- [7] C. Fragouli and E. Soljanin, "Information flow decomposition for network coding," *IEEE Trans. Inform. Theory*, vol. 52, no. 3, pp. 829–848, 2006.
- [8] K. Bhattad, N. Ratnakar, R. Koetter, and K. R. Narayanan, "Minimal network coding for multicast," in *Proc. IEEE ISIT '05*.
- [9] R. Elbaum and M. Sidi, "Topological design of local-area networks using genetic algorithms," *IEEE/ACM Trans. Networking*, vol. 4, no. 5, pp. 766–778, 1996.
- [10] B. Dengiza, F. Altıparmak, and A. E. Smith, "Efficient optimization of all-terminal reliable networks, using an evolutionary approach," *IEEE Trans. Rel.*, vol. 46, no. 1, pp. 18–26, 1997.
- [11] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1996.
- [12] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer*, vol. 27, no. 6, pp. 17–26, 1994.
- [13] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. Networking*, vol. 11, no. 5, pp. 782–795, 2003.
- [14] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *J. Symb. Comput.*, vol. 9, no. 3, pp. 251–280, 1990.
- [15] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.
- [16] G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller, "The gambler's ruin problem, genetic algorithms, and the sizing of populations," *Evolutionary Computation*, vol. 7, no. 3, pp. 231–253, 1999.
- [17] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, "Polynomial time algorithms for multicast network code construction," *IEEE Trans. Inform. Theory*, vol. 51, no. 6, pp. 1973–1982, 2005.
- [18] T. Ho, B. Leong, R. Koetter, and M. Médard, "Distributed asynchronous algorithms for multicast network coding," in *Proc. NetCod*, 2005.
- [19] T. Ho, M. Médard, J. Shi, M. Effros, and D. R. Karger, "On randomized network coding," in *Proc. Annual Allerton Conference on Communication, Control, and Computing*, 2003.
- [20] E. Erez and M. Feder, "Efficient network codes for cyclic networks," in *Proc. ISIT*, 2005.
- [21] D. S. Lun, M. Médard, T. Ho, and R. Koetter, "Network coding with a cost criterion," MIT-LIDS, Tech. Rep. P-2584, 2004.
- [22] G. Melançon and F. Philippe, "Generating connected acyclic digraphs uniformly at random," *Inf. Process. Lett.*, vol. 90, no. 4, pp. 209–213, 2004.
- [23] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proc. ACM/SIGCOMM '02*, pp. 133–145.