Title: On Minimizing Coding Operations in Network Coding Based Multicast: An Evolutionary Algorithm

Authors: Huanlai Xing (corresponding author)[1], Rong Qu[2], Lin Bai[3], Yuefeng Ji[3]

Affiliation 1: School of Information Science and Technology, Southwest Jiaotong University
Address: 111, North 1st Section, 2nd Ring Rd, Chengdu 610031, P.R.China

Affiliation 2: ASAP Group, School of Computer Science, The University of Nottingham
Address: School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, United Kingdom

Affiliation 3: State Key Laboratory of Information Photonics and Optical Communications, Beijing University of Posts and Telecommunications
Address: 10 XITUCHENG Rd, Haidian District, Beijing 100876, P.R.China

Phone: +86 18780272961 (corresponding author)
Email: hxx@home.swjtu.edu.cn (corresponding author)

***Abstract*:**

In telecommunications networks, to enable a valid data transmission based on network coding, any intermediate node within a given network is allowed, if necessary, to perform coding operations. The more coding operations needed, the more coding resources consumed and thus the more computational overhead and transmission delay incurred.

This paper investigates an efficient evolutionary algorithm to minimize the amount of coding operations required in network coding based multicast. Based on genetic algorithms, we adapt two extensions in the proposed evolutionary algorithm, namely a new crossover operator and a neighbourhood search operator, to effectively solve the highly complex problem being concerned. The new crossover is based on logic OR operations to each pair of selected parent individuals and the resulting offspring are more likely to become feasible. The aim of this operator is to intensify the search in regions with plenty of feasible individuals. The neighbourhood search consists of two moves which are based on greedy link removal and path reconstruction, respectively. Due to the specific problem feature, it is possible that each feasible individual corresponds to a number of, rather than a single, valid network coding based routing subgraphs. The neighbourhood search is applied to each feasible individual to find a fitter routing subgraph that consumes less coding resource. This operator not only improves solution quality but also accelerates the convergence. Experiments have been carried out on a number of fixed and randomly generated benchmark networks. The results demonstrate that with the two extensions, our evolutionary algorithm is effective and outperforms a number of state-of-the-art algorithms in terms of the ability of finding optimal solutions.

Keywords:

Evolutionary computation, multicast, network coding, telecommunications

1. Introduction

Multicast is a one-to-many communication technique that simultaneously delivers information from the source to a group of destinations (receivers) within the same network so that in a single transmission any receiver of the group is able to obtain the original information sent from the source [1]. With the emergence of increasingly more multimedia applications such as video conferencing and IPTV, multicast has become one of the key supporting technologies in modern communication networks [2,3].

Network coding, as a new communication paradigm, has received an increasing amount of research attention since 2000 [4]. It is superior to the traditional routing protocol in many aspects, including an increased multicast throughput, a balanced network payload, transmitting energy savings, and so on [5-8]. It is well known that traditional routing transmits information in the same way as fluids share pipes [8]. Different data streams (data information) transmitted share the limited network resources, however, are separately processed at the network-layer. In traditional routing, each intermediate node simply forwards the incoming data packets to its downstream node(s), adopting *store-and-forward* data forwarding scheme. However, the theoretical maximum throughput of a multicast may not be achieved by using such scheme [4,5]. In contrast, with *code-and-forward* data forwarding scheme at the network-layer, network coding allows any intermediate node to perform mathematical operations to data packets received from different incoming links if necessary, always achieving the maximized multicast throughput according to the MAX-FLOW MIN-CUT theorem [5].

In the current literature, the majority of the research in network coding assumes that coding operations should be carried out at all coding-possible nodes. However, to obtain an expected throughput, coding may only be necessary at a subset of these nodes [9-11]. Since coding operation consumes computational overhead and increases data processing complexity, it is of great interest to minimize the amount of coding operations. Such problem is proven to be NP-Hard [9-11].

Evolutionary algorithms (EAs) are a class of population-based meta-heuristic optimization techniques based on the mechanics of natural selection and reproduction [12]. EAs often perform well on approximating solutions to many types of optimization problems because there is no need for users to make any assumption about the underlying fitness landscape. It is well known that genetic algorithm (GA) is one of the mostly investigated evolutionary algorithms and has been successfully and widely applied in operations research, engineering, economics, and so on. Although GA is a general bio-inspired algorithm, care should be taken in designing effective operators to the given optimization problem to avoid potential problems such as pre-maturity and slow convergence. In addition, the local exploitation ability in standard GAs cannot usually be fully achieved, motivating the advanced research in developing hybrid evolutionary algorithms including memetic algorithms [13,14].

In this paper, we present an efficient evolutionary algorithm with two new schemes to minimize the amount of coding operations required in network coding based multicast. In the first scheme, a new crossover operator is devised to intensify the exploration in the regions of solution space where the majority of individuals are feasible. Different from the standard crossover that swaps a subset of the selected parent individuals, the new crossover performs the logic OR operation to the selected parent individuals and thus can effectively produce feasible individuals. Our experimental results show that the global exploration has been improved by integrating the new crossover and simple mutation compared with using the uniform crossover and simple mutation. In the problem concerned, each feasible individual may correspond to several valid network coding based routing schemes which consume different amount of coding resources. We design a neighborhood search operator, by using two moves, to select a better routing scheme for the feasible individual. The experimental results also show that, with the neighbourhood search and the new crossover, the proposed algorithm is able to find an optimal solution in all instances, including *x*-copies and random networks.

2. Problem Formulation and Related Work

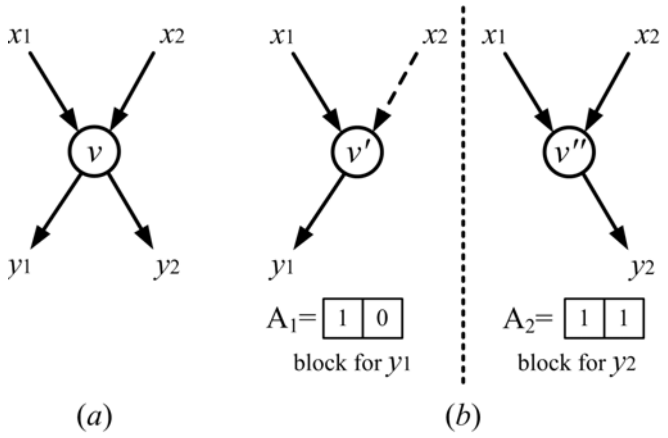
2.1 Problem Formulation

A communication network can be modeled as a directed graph $G = (V, E)$, where V and E denote the set of nodes and links, respectively [5]. We assume that each link $e \in E$ has a unit capacity. Only integer flows are allowed in G so a link is either idle or occupied by a flow of unit rate [10,11]. In this paper, we only consider the multicasting function in telecommunications networks by using the network coding technology. A single-source network coding based multicast scenario can be defined as a 4-tuple set (G, s, T, R), where the information needs to be transmitted at data rate R from the source node $s \in V$ to a set of sinks $T = \{t_1,\ldots,t_d\} \subset V$ in the graph G (V, E). The data rate R is achievable if there is a transmission scheme that enables each sink t_k, $k = 1, \ldots, d$, to receive the information at rate R [10,11]. As each link has a unit capacity, any single path connecting s and t_k ($k = 1, \ldots, d$) has a unit capacity. If we manage to set up R link-disjoint paths from s to each receiver t_k, i.e. $P_1(s, t_k),\ldots,P_R(s, t_k)$, data rate R is achievable.

We refer to a connected subgraph in G, including s and all sinks in T, as a network coding based routing subgraph $G_{s\rightarrow T}$ if the data rate from s to each $t_k \in T$ is of R units within this subgraph. In a routing subgraph, there are R link-disjoint paths from s to each receiver. To find a routing subgraph, we first find R link-disjoint paths for each $t_k \in T$ from G, i.e. $P_1(s, t_k),\ldots,P_R(s, t_k)$. There are $R\cdot d$ paths in total. We then mark those links and nodes in G which are being occupied by at least one of the $R\cdot d$ paths. The routing subgraph is the union of marked nodes and links in G. Note that in a routing subgraph, paths to the same receiver never join together since they are link-disjoint. Hence, only those paths to different receivers are possible to be involved in coding. In routing subgraph, a node is called *coding node* if data streams from different incoming links of it are recombined together. An outgoing link of a coding node is called a *coding link* if the coded data stream is sent out via this link.

In a network G, no coding occurs at a node with single incoming link. We refer to a non-sink intermediate node with multiple incoming links as a *merging node* [10,11]. Consider a merging node v with $In(v)$ incoming links and $Out(v)$ outgoing links, where $In(v) \geq 2$ and $Out(v) \geq 1$. We use a decision variable $\mathrm{A}_j = \{a_{ij} \mid i = 1,\ldots, In(v), j = 1,\ldots,Out(v)\}$, where A_j is a block of length $In(v)$, to represent the information of how many incoming links of v contributing to the output over the j-th outgoing link of v [10,11]. For each $i \in \{1,\ldots, In(v)\}$ and each $j \in \{1,\ldots, Out(v)\}$, if the information from the i-th incoming link contributes to the coded output over the j-th outgoing link, $a_{ij} = 1$, otherwise $a_{ij} = 0$. We refer to the states of '1' and '0' as *active* and *inactive* states, respectively [10,11]. Network coding is performed at node v and the coded output is transmitted over link j only if at least two incoming link states of link j are active simultaneously. Fig.1 shows an example of a merging node and a possible set of link states, respectively [15]. The individual representation in our evolutionary algorithm consists of blocks of all potential coding links (refer to section

3). We refer to each outgoing link of a merging node as a *potential coding link*. To determine if a potential coding link serves as a coding link, we need to check if the information via this link is dependent on a number of incoming links of the merging node. We only consider linear network coding which is sufficient for multicast [5].



(a) Merging node v (b) Two blocks for outgoing links

Fig.1 Node v with two incoming and two outgoing links, described by input variables $\mathbf{A}_1 = (a_{11}, a_{21})$ and $\mathbf{A}_2 = (a_{12}, a_{22})$.

The number of coding links (rather than coding nodes) is more accurate to indicate the total amount of coding operations involved [16]. We hereafter investigate on finding a routing subgraph $G_{s\rightarrow T}$ with minimized coding links. The following notations are defined:

σ_{ij} :	a binary variable associated with the j-th outgoing link of the i-th merging node, $i = 1,\ldots,M$, $j = 1,\ldots,Z_i$, where M is the total number of merging nodes and the i-th node has Z_i outgoing links. $\sigma_{ij} = 1$ if the j-th outgoing link of the i-th node serves as a coding link; $\sigma_{ij} = 0$ otherwise.
$\lambda(s, t_k)$:	the achievable rate between s and t_k in $G_{s\rightarrow T}$
$p_i(s, t_k)$:	the i-th link-disjoint path found between s and t_k in $G_{s\rightarrow T}$, $i = 1,2,\ldots,R$.
$W_i(s, t_k)$:	the link set of $P_i(s, t_k)$, i.e. $W_i(s, t_k) = \{e \mid e \in P_i(s, t_k)\}$.

Based on the above notations, we define in this paper the problem of finding a routing subgraph $G_{s\rightarrow T}$ where the number of coding links, $\Phi(G_{s\rightarrow T})$, is minimized while the desired data rate R is achieved, shown as follows:

Minimize: $$\Phi(G_{s\rightarrow T}) = \sum_{i=1}^{M}\sum_{j=1}^{Z_i}\sigma_{ij} \quad (1)$$

Subject to: $$\lambda(s, t_k) = R,\ \forall\ t_k \in T \quad (2)$$

$$W_i(s,t_k) \cap W_j(s,t_k) = \emptyset, \quad \forall i,j \in \{1,\ldots,R\}, i \neq j, t_k \in T \tag{3}$$

Objective (1) defines our problem as to find a routing subgraph $G_{s\to T}$ with the minimum number of coding links; Constraint (2) defines that in $G_{s\to T}$, the achievable data rate between s and each sink is R; Constraint (3) indicates that for an arbitrary t_k the R constructed paths $P_i(s, t_k)$, $i = 1,\ldots,R$, have no common link.

2.2 Related Work

In the past decade, a considerable amount of research has been dedicated to encoding-and-decoding approaches for network coding based data transmission [5,17-20]. However, the minimization on coding resources in network coding based multicast has not received enough attention.

The first seminal work on the minimization problem concerns two greedy approaches proposed, respectively. In [21], information flows are decomposed into a number of subtrees, where the same information is transported in the same subtree. Besides, a greedy algorithm is presented to construct a subtree graph involving minimal coding operations while supporting a valid network coding based data transmission. In [16], a given network is transformed into a new network first, where the degree of each node is at most three. After that, the authors remove links which make no contribution to the achievable data rate from the new graph to reduce the coding operations involved. However, the performance of the above two algorithms depends on the traversal order of links, e.g. an inappropriate link traversal order may significantly weaken the optimization performance.

From then on, a number of evolutionary algorithms (EAs) were proposed. Kim *et al* used genetic algorithms (GAs) to optimize the required network coding resource [9-11]. In 2006, they proposed a GA on an algebraic framework [9]. Given a network *G*, they construct the corresponding labeled line graph *G'* by using the information flow decomposition method [21]. Then each link $e \in G'$ is associated with a link coefficient. All the coefficients result into a specific solution based on which the consumed network coding resource can be calculated. The algebraic framework is the first method which maps the network coding problem to a GA framework. However, this GA is only applicable to acyclic networks. After that, Kim *et al* extend their work in [9] to a distributed GA to significantly reduce the computational time [10]. Besides, a graph decomposition method is proposed for both acyclic and cyclic networks. It decomposes each merging node in *G* into a number of auxiliary nodes to show explicitly how a flow passes through the merging node. This method then becomes a popular way to map the network coding problem to an EA framework. After that, Kim *et al* compare and analyse GAs with two different genotype encoding approaches, i.e. the binary link state (BLS) and the block transmission state (BTS), and their associated genetic operators [11]. Compared with BLS encoding, BTS encoding has a much smaller solution space and usually leads to better solutions. Besides, their GA-based algorithms perform outstandingly better than the two greedy algorithms in [16] and [21] in terms of the best solutions achieved. However, as we observed in this paper, the major genetic operator, i.e. the uniform crossover, has a limited contribution to produce promising solutions. Recently, a path-oriented encoding, which leads to a completely feasible search space, is also designed for EAs for tackling the network coding problem [22]. Nevertheless, this encoding is complex (e.g. two-dimensional) and its size grows with the number of receivers. Hence EAs with the path-oriented encoding are limited to applications with relatively small multicast group.

In addition to GAs, estimation of distribution algorithms (EDAs) are also used to solve the problem concerned. Two quantum-inspired evolutionary algorithms (QEAs) are proposed to minimize the coding operations involved [15,23]. QEAs maintain a population of quantum-bit individuals, each representing a linear superposition of all solutions in the search space. Simulation results demonstrate that the improved QEA outperforms a standard GA in some instances, however at the cost of additional computational time. Xing and Qu present a population based incremental learning algorithm (PBIL) which integrates GA with competitive learning [24]. PBIL maintains a real-valued probability vector which, when sampled, generates better solutions with a higher probability during the evolution. With a restart scheme, the PBIL usually finds decent solutions, however, consuming a considerable amount of computational time. Another PBIL is introduced to mainly consider how to adapt network coding in delay sensitive applications, where delay bound is concerned [25]. Later on, a compact genetic algorithm is proposed for minimizing the amount of coding operations in the network coding based multicast, where at each generation only one solution is sampled from the probability vector. It competes with the best solution of previous generations and the winner is used to update the probability vector [26].

Some researchers study the minimum-cost network coding problem using evolutionary approaches with entropy-based evaluation relaxation techniques in order to reduce the computational cost incurred during the evolution [27,28]. By making use of the inherent randomness feature of the individuals, the proposed EAs can rapidly recognize promising solutions with much fewer individuals to be evaluated. Recently, a multi-objective EA is adopted to balance the consumed resource and user experience during the network coding based multicast [29].

In general, standard EAs may be directly applied to a range of optimization problems. However, in some applications of EAs, its genetic operators are too general to warrantee an efficient and effective evolution. Our motivation in this paper is to investigate an effective EA with problem-specific genetic operators devised to the problem concerned based on the domain knowledge and characteristics obtained from the problem.

3. The Proposed Algorithm

Our evolutionary algorithm is based on GA and extended by employing an OR-based crossover and a neighborhood search operator. This section first introduces the individual representation and fitness evaluation. After that, the two extensions are described in details. Finally, we provide the overall procedure of our algorithm.

3.1 Individual Representation and Evaluation

When designing EAs, chromosome representation is one of the most important issues. In this paper, we adopt the binary link state (BLS) encoding to represent solutions, because it has been successfully used in a number of network coding resource minimization problems [10,11,24-26].

As mentioned in section 2, only merging nodes can perform coding if necessary. To explicitly show all possible ways of how information flows via a particular merging node, the graph decomposition method is employed to decompose each merging node in G into a number of nodes connected by links [10,11]. As a result, a secondary graph G_D is created. The detailed procedure is shown below. For the i-th merging node, let $In(i)$ be the number of incoming links and $Out(i)$ be the number of outgoing links, respectively. The original i-th merging node is decomposed into two sets of nodes: (1) $In(i)$ nodes, $u_1,\ldots,u_{In(i)}$, referred to as incoming auxiliary nodes, and (2) $Out(i)$ nodes, $w_1,\ldots,w_{Out(i)}$, referred to as outgoing auxiliary nodes. The j-th incoming link of the i-th original merging node is redirected to node u_j; and the k-th outgoing link of the i-th merging node is redirected to node w_k. Besides, a directed link $e(u_j, w_k)$ is inserted between u_j and w_k, $j = 1,\ldots,In(i)$, $k = 1,\ldots,Out(i)$. The graph decomposition method is widely adopted in the literature [10,11,24-26].

In the BLS encoding, an individual (solution) $\mathbf{X} = \{x_1, x_2,\ldots, x_m\}$ is represented by a string of binary bits, where each bit x_i is associated with one of the newly inserted links between auxiliary nodes, e.g. $e(u_j, w_k)$ in G_D. Value ‘1’ at bit x_i means its corresponding link exists in G_D, and value ‘0’ otherwise. Hence, each individual corresponds to an explicit secondary graph G_D which may or may not support a valid network coding based multicast routing solution.

To evaluate a given individual $\mathbf{X}$, we first check if $\mathbf{X}$ is feasible. Each individual corresponds to a unique secondary graph G_D. We compute the max-flow between the source s and an arbitrary receiver $t_k \in T$ in G_D using max-flow algorithm introduced in [30]. As mentioned in section 2, we assume each link in G has a unit capacity. The max-flow between s and t_k is thus equivalent to the number of link-disjoint paths between s and t_k found by the max-flow algorithm. If all d max-flows are at least R, where d is the number of receivers, rate R is achievable and the individual $\mathbf{X}$ is feasible. Otherwise, $\mathbf{X}$ is infeasible.

For each infeasible individual $\mathbf{X}$, we set $f(\mathbf{X}) = \Psi$, where $f(\mathbf{X})$ is the fitness value of $\mathbf{X}$ and Ψ is a sufficiently large integer (in this paper, $\Psi = 50$). If $\mathbf{X}$ is feasible, we first find a routing subgraph $G_{s\to T}$ from G_D and then calculate its fitness. For each sink $t_k \in T$, we select R paths from the obtained link-disjoint paths from s to t_k. We therefore obtain in total $R\cdot d$ paths, e.g. $p_i(s, t_k)$, $i = 1,\ldots,R$, $k = 1,\ldots,d$. $G_{s\to T}$ is a subgraph of G_D that only contains all these $R\cdot d$ paths. In $G_{s\to T}$, coding operation happens at those outgoing auxiliary nodes with two or more incoming links. The fitness value $f(\mathbf{X})$ is set to the number of coding links in $G_{s\to T}$.

Fig.2 shows an example of the graph decomposition and an obtained routing subgraph. Fig.2(a) is an original network where source s expects to transmit the information to two sinks t_1 and t_2 at a data rate of 2, i.e. $R = 2$. The graph has two merging nodes, i.e. v_1 and v_2. By using the graph decomposition method, the original graph is transformed into a decomposed graph, as shown in Fig.2(b). Nodes v_1 and v_2

are decomposed into two groups of auxiliary nodes with newly inserted links e_1, …, e_8. We associate the i-th bit of an individual with link e_i, i = 1,…,8. Given an individual '01110111', its corresponding secondary graph G_D is shown in Fig.2(c). Based on G_D, a routing subgraph $G_{s\to T}$ with only one coding node $w2$ is obtained, as shown in Fig.2(d).

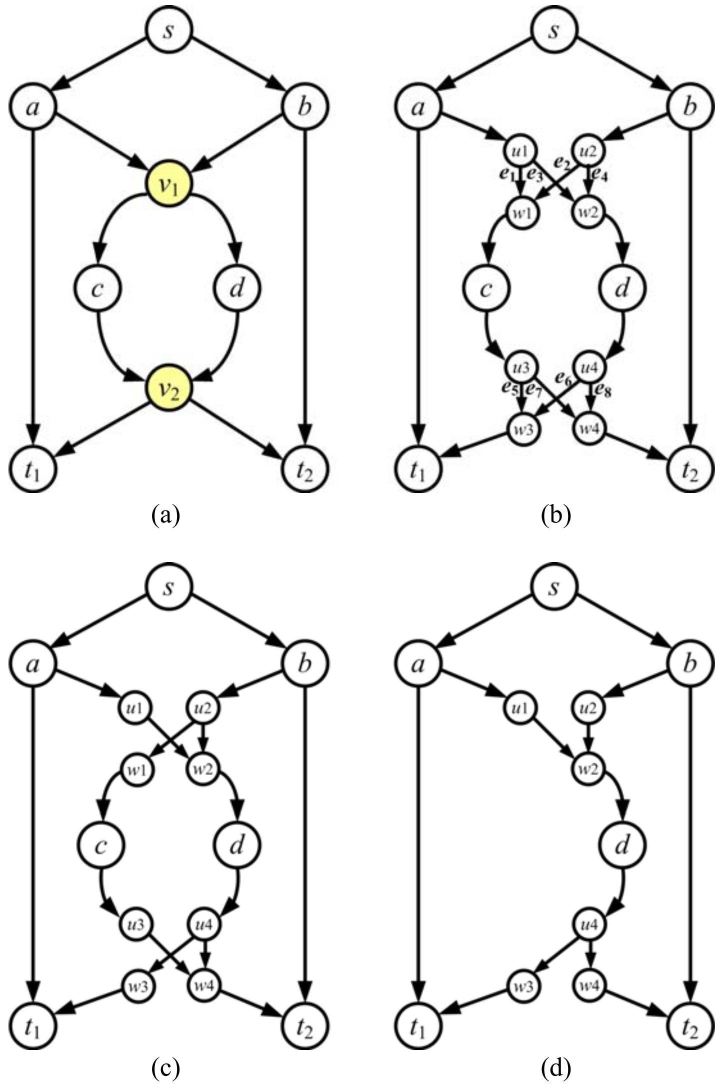


Fig.2 An example of the graph decomposition and an obtained routing subgraph (a) The original network (b) The decomposed network (c) The corresponding G_D (d) The obtained routing subgraph

3.2 The OR-based Crossover (ORX)

As the problem concerned is highly constrained, infeasible solutions take up a considerable part of the solution space. The performance of GA could be seriously weakened if it cannot generate sufficient feasible individuals.

Crossover is regarded as a major genetic operator [12]. To help GA to achieve a decent performance, when applied to a specific problem, crossover needs to be defined properly. Kim et al. adopt uniform crossover (UX) where each allele is exchanged between the selected pair of parent individuals with a given swapping probability (mixing ratio) typically set to 0.5 [12]. However, this crossover is too general and may not be appropriate to create feasible offspring for the problem concerned in this paper. Kim *et al* noticed the lack of feasible individuals in GA and inserted an all-one vector into the initial population to make sure that their algorithm starts with at least one feasible individual [10,11].

Concerning the aforementioned individual representation, it can be seen that the more 1s in an individual, the more newly inserted links exist in the corresponding G_D, and thus the higher possibility this individual is feasible. This is the reason why Kim *et al* use an all-one vector to warrantee at least one feasible individual in the evolution.

Motivated by the idea that individuals containing more 1s have higher probabilities to be feasible, we propose a novel crossover operator based on the logic OR operation. Similar to the single-point crossover, our crossover is performed at a crossover probability p_c, and at a crossover point to the selected pairs of individuals. However, the way the two offspring inherit genes from their parents in our crossover is based on the bitwise logic OR operation. Assume the length of each individual is L. The steps of the proposed crossover are as follows:

1) Given a population $\{\mathbf{X}_1, \mathbf{X}_2,\ldots, \mathbf{X}_{2N}\}$, the $2N$ individuals are randomly divided into N pairs, where N is an integer and $2N$ is the population size.

2) For the i-th pair $(\mathbf{X}_j, \mathbf{X}_k)$, we generate a random number r_i and compare it with the crossover probability p_c. If $r_i \leq p_c$, we choose the i-th pair to perform crossover and go to step 3; otherwise, the i-th pair remains in the population.

3) Randomly select a crossover point from $\{2,\ldots,L\}$ for the i-th pair, e.g. ℓ. We denote by $\mathbf{X}(m:n)$ the substring from the m-th bit to the n-th bit of an individual $\mathbf{X}$. Then, the crossover to the i-th pair $(\mathbf{X}_j, \mathbf{X}_k)$ is performed as follows. The first offspring directly inherits $\mathbf{X}_j(1:\ell\text{-}1)$ as its 1 to ℓ-1 bits. Its ℓ to L bits are obtained by performing a bit-wise logic OR operation to $\mathbf{X}_j(\ell:L)$ and $\mathbf{X}_k(\ell:L)$. Similarly, the second offspring directly inherits $\mathbf{X}_k(\ell:L)$ as its ℓ to L bits. The 1 to ℓ-1 bits of the second offspring are obtained by performing a bitwise logic OR operation to $\mathbf{X}_j(1:\ell\text{-}1)$ and $\mathbf{X}_k(1:\ell\text{-}1)$.

4) After crossover, the population is updated by replacing parent individuals with their offspring.

To show how the OR-based crossover (ORX) is performed, we use the same problem in Fig.2(*a*) as an example. The *i*-th bit of an individual is associated with the *i*-th inserted link, e_i, $i = 1,\ldots,8$ (see Fig.2(*b*)). Suppose a pair of individuals, parent1 = '11001001' and parent2 = '00110110', are chosen for crossover. It is easy to see that these two individuals are both infeasible (for feasibility verification, see section 3.1). Assume the crossover point $\ell = 5$. Fig.3 shows an example of the OR-based crossover on the two parent individuals to generate the two offspring, i.e. '11001111' and '11110110', which are both feasible. It is obvious that the bitwise logic OR operator can help to increase the proportion of 1s in each offspring, while preserving characteristics of the parent individuals. The number of feasible individuals is potentially increasing after the crossover based on the OR operator.

The OR-based crossover (ORX), on the one hand, is designed to intensify the search in a region with plenty of feasible individuals. It should usually find such a region within a short number of generations due to that the logic OR helps to produce increasingly more feasible individuals. On the other hand, however, different from the uniform crossover, the OR-based crossover will not help much to explore new regions in the solution space once feasible individuals are sufficient in the population. What the OR-based crossover does is to iteratively confine the search in a smaller and smaller region, and thus leading to pre-maturity. We therefore also employ mutation in our EA to diversify the population and assist local search (see section 3.4).

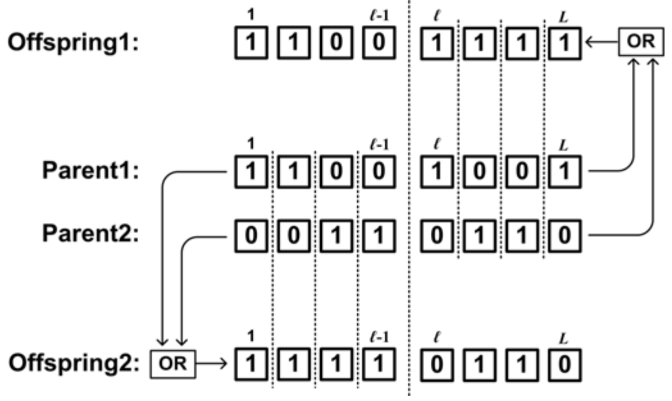


Fig.3 An example of generating offspring using the OR-based crossover

3.3 The Neighbourhood Search Operator

As mentioned in section 3.1, each feasible individual corresponds to a secondary graph G_D based on which a routing subgraph $G_{s\to T}$ can be found. However, one issue to be addressed is that, it is possible that more than one feasible routing subgraph exists in the given G_D, although by using the method in fitness evaluation we can only obtain one. For the example problem in Fig.2(*a*), the corresponding secondary graph and two possible feasible routing subgraphs obtained for an individual '11011110' are shown in Fig.4, where the subgraph in Fig.4(*c*) has less coding links than in Fig.4(*b*). The better the routing subgraph found (i.e. the subgraph requires less coding operations), the higher the quality of the corresponding solution to the problem, and the faster the convergence of the algorithm. So, an interesting research question arises: shall we explore each secondary graph of each feasible individual to find better routing subgraph?

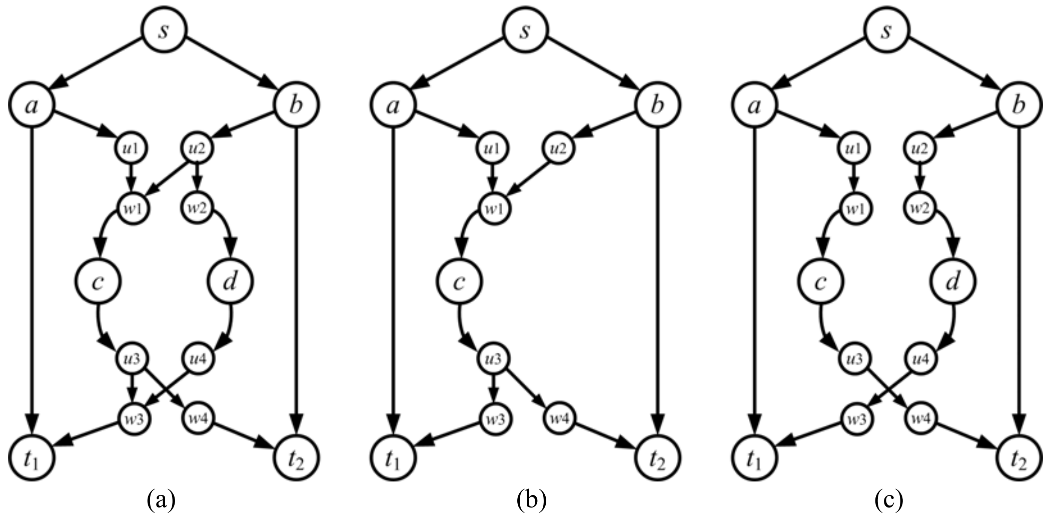


Fig.4 An example of different routing subgraphs obtained from a given secondary graph (a) The corresponding G_D (b) A feasible routing subgraph (c) Another feasible routing subgraph

Inspired by the issue and question above, we design a neighborhood search operator to enhance the global exploration of our algorithm. Given a feasible individual **X**, we denote by $G_{s\to T}(start)$ the routing subgraph obtained by fitness evaluation (see section 3.1). Starting from $G_{s\to T}(start)$, our neighborhood search, by using two moves, explores G_D to find a new routing subgraph which requires less coding operations. The two moves, i.e. the link removal based move (M1) and the path reconstruction based move (M2), are cascaded to find better routing subgraphs. We denote the incumbent routing subgraph found by M1 and M2 as $G_{s\to T}(M1)$ and $G_{s\to T}(M2)$, respectively. We hereafter call a routing subgraph as a coding-free routing subgraph if this subgraph does not require any coding operation.

M1 aims to remove those incoming links in G_D which contribute to coding operations yet are redundant to the data rate R. Fig.5 shows the procedure of M1.

```
1) Set $G_{temp} = G_D$ and $G_{s\to T}(M1) = G_{s\to T}(start)$.
2) If $G_{s\to T}(M1)$ is coding-free, stop the procedure and output $G_{s\to T}(M1)$.
3) Mark all coding nodes in $G_{s\to T}(M1)$ as untraversed.
4) Randomly select an untraversed coding node in $G_{s\to T}(M1)$, e.g. node $n_c$. Assume $n_c$ has
   $In(n_c)$ incoming links in $G_{s\to T}(M1)$, where $In(n_c) \geq 2$. Number these incoming links in
   any order.
5) For $i = 1$ to $In(n_c)$
   a) Delete the $i$-th incoming link of $n_c$ from $G_{temp}$.
   b) Check if a new routing subgraph $G_{s\to T}(new)$ can be obtained from $G_{temp}$ by the fitness
      evaluation in section 3.1. If $G_{s\to T}(new)$ is found and $\Phi(G_{s\to T}(new)) \leq \Phi(G_{s\to T}(M1))$,
      set $G_{s\to T}(M1) = G_{s\to T}(new)$ and go to step 2; Otherwise, reinsert the $i$-th incoming
      link to $G_{temp}$.
6) Mark $n_c$ as a traversed coding node in $G_{s\to T}(M1)$. If all coding nodes in $G_{s\to T}(M1)$ are
   traversed, stop the procedure and output $G_{s\to T}(M1)$; otherwise, go to step 4.
```
Fig.5 Procedure of the link removal based move (M1)

After M1, the routing subgraph obtained, i.e. $G_{s\to T}(M1)$, may be substantially improved compared with $G_{s\to T}(start)$. However, it is still possible that some coding nodes remain in $G_{s\to T}(M1)$. This is because M1 operates in a greedy fashion. Although it is simple to implement, M1's performance may vary with the structure of $G_D(start)$ and different traversal order of incoming links. Especially when $G_D(start)$ does not contain enough incoming links for deletion, M1 may not be able to eliminate all coding nodes in $G_{s\to T}(M1)$. Hence, using M1 does not guarantee to result in a coding-free routing subgraph. To optimize the structure of $G_{s\to T}(M1)$, we design M2 to further reduce the number of coding nodes. Set $G_{s\to T}(M2) = G_{s\to T}(M1)$, and then M2 starts from $G_{s\to T}(M2)$.

As mentioned in section 2, each routing subgraph consists of $R\cdot d$ paths, where R is the data rate and d is the number of sinks. Only those paths to different sinks may contribute to a coding operation. M2 aims to avoid each coding node by reconstructing some paths in $G_{s\to T}(M2)$ that contribute to the coding at the node.

For each coding node, path reconstruction includes four steps: (1) Determine which paths in $G_{s\to T}(M2)$ are involved in the coding at the node. (2) For each involved path, select the *start* and *end* points of a subpath that needs to be reconstructed. (3) Find an alternative subpath between the start and end points that do not contribute to any coding node. (4) Update the structure of $G_{s\to T}(M2)$.

For convenience, we number all paths in $G_{s\to T}(M2)$, i.e. $p_i(s,t_k)$, $i = 1, 2, \ldots, R$, $k = 1,2,\ldots,d$, as path 1, 2, …, $R\cdot d$. We identify how many paths occupy a link $e\in G_{s\to T}(M2)$, and label each e with the indices of paths that occupy e. For example, we number the four paths of the routing subgraph in Fig.4(b), where $R = 2$ and $d = 2$, i.e.

$p_1(s,t_1) = s\to a\to t_1$;

$p_2(s,t_1) = s\to b\to u_2\to w_1\to c\to u_3\to w_3\to t_1$;

$p_1(s,t_2) = s\to a\to u_1\to w_1\to c\to u_3\to w_4\to t_2$;

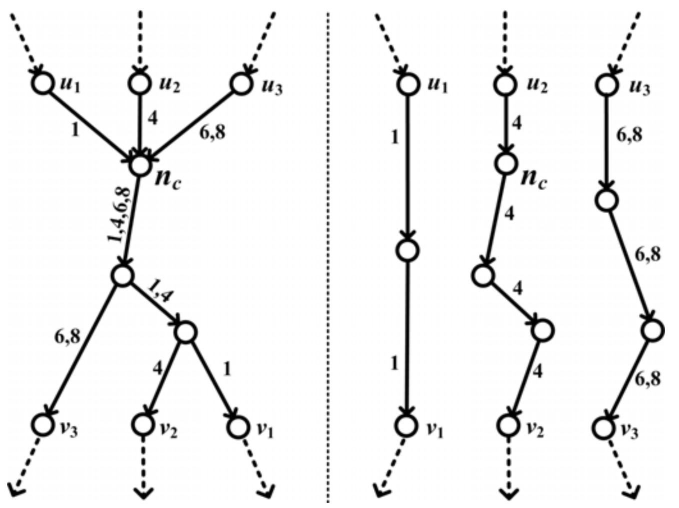
$p_2(s,t_2) = s\to b\to t_2$;

as path 1, 2, 3 and 4, respectively, with labels on each link as shown in Table 1. We hereafter use paths $1,2,\ldots,R\cdot d$ to represent all paths in $G_{s\to T}(M2)$. Let $SubPath(i,u,v)$ denote the subpath of path i from node u to node v.

Table 1 Links and Their Path Labels in Fig.4(b)

Link	Label	Link	Label	Link	Label
$s\to a$	1,3	$b\to u_2$	2	$u_3\to w_3$	2
$s\to b$	2,4	$u_1\to w_1$	3	$u_3\to w_4$	3
$a\to t_1$	1	$u_2\to w_1$	2	$w_3\to t_1$	2
$b\to t_2$	4	$w_1\to c$	2,3	$w_4\to t_2$	3
$a\to u_1$	3	$c\to u_3$	2,3		

Fig.6 illustrates an example of path reconstruction within $G_{s\to T}(M2)$ by using M2. Note that number(s) attached to each link is the path label of the link. We see that four subpaths, i.e. $SubPath(1,u_1,v_1)$, $SubPath(4,u_2,v_2)$, $SubPath(6,u_3,v_3)$ and $SubPath(8,u_3,v_3)$, form the coding node n_c. Due to $SubPath(6,u_3,v_3) = SubPath(8,u_3,v_3)$, M2 treats them the same when reconstructing subpath from u_3 to v_3. After path reconstruction, subpaths $SubPath(1,u_1,v_1)$, $SubPath(6,u_3,v_3)$ and $SubPath(8,u_3,v_3)$ are reconstructed and the original coding operation at node n_c is avoided. Beside, $G_{s\to T}(M2)$ with these new subpaths is still feasible as each sink can still receive a data rate of R. In this way, M2 can reduce the number of coding nodes in $G_{s\to T}(M2)$ while keeping the feasibility of $G_{s\to T}(M2)$.



(a) Before reconstruction (b) After reconstruction

Fig.6 An example of path reconstruction. (a) target subpath in $G_{s\to T}(M1)$ that forms a coding node n_c (b) subgraph obtained after reconstruction.

The determination of which paths contribute to a given coding node n_c is easy. To identify them, we only need to check path labels of incoming links of n_c. Some incoming links of n_c are occupied by multiple paths (e.g. $u_3 \rightarrow n_c$ in Fig.6(a)) while others are occupied by single path (e.g. $u_1 \rightarrow n_c$ in Fig.6(a)).

The following steps are to determine the *start* point and *end* point of a subpath for path reconstruction (e.g. u_1 and v_1 in Fig.6):

- For any incoming link of n_c occupied by multiple paths, we randomly choose one path (e.g. PathA) and check the upstream and downstream nodes of n_c along the path (PathA). Given a node n along PathA, we denote the incoming link of n along PathA by $e_{in}(n,\text{PathA})$ and the outgoing link of n along PathA by $e_{out}(n,\text{PathA})$. Let $\Lambda(e_{in}(n,\text{PathA}))$ and $\Lambda(e_{out}(n,\text{PathA}))$ denote the labels of $e_{in}(n,\text{PathA})$ and $e_{out}(n,\text{PathA})$, respectively. When searching upstream, we refer to a node n_1 as a start point for path reconstruction if n_1 satisfies the following conditions: $\Lambda(e_{out}(n_1,\text{PathA})) = \Lambda(e_{in}(n_c,\text{PathA}))$ and $\Lambda(e_{in}(n_1,\text{PathA})) \neq \Lambda(e_{in}(n_c,\text{PathA}))$. We stop the upstream search once such n_1 is found. If n_1 cannot be found from the search, we set the source s as the *start* point. Similarly, when searching downstream, we refer to a node n_2 as an end point for path reconstruction if n_2 satisfies the following conditions: $\Lambda(e_{in}(n_2,\text{PathA})) = \Lambda(e_{in}(n_c,\text{PathA}))$ and $\Lambda(e_{out}(n_2,\text{PathA})) \neq \Lambda(e_{in}(n_c,\text{PathA}))$. We stop the downstream search once n_2 is found. Otherwise, we set the sink of PathA as the end point. Based on the pair of *start* and *end* points, the subpath occupied by multiple paths between them needs to be reconstructed to avoid the coding at n_c. It is not difficult to understand that the above subpath is the maximal length subpath the multiple paths share in common. Such subpath enables us to treat multiple paths as a whole when reconstructing subpaths.

- For any incoming link occupied by a single path, we search along this single path and use the above same rule to determine a pair of start and end points for path reconstruction.

To remove the coding operation at n_c from $G_{s\rightarrow T}(M2)$, we need to find alternative subpaths (from start point to end point) to replace the conflicting subpaths (see Fig.6(*b*)).

Assume n_c has $In(n_c)$ incoming links in $G_{s\rightarrow T}(M2)$, where $In(n_c) \geq 2$. We have $In(n_c)$ pairs of start and end points. Let u_i and v_i, $i = 1, 2, \ldots, h$, be the i-th pair of start point and end point, respectively. Among these pairs of nodes, we can avoid the coding at n_c if we successfully reconstruct $In(n_c) - 1$ subpaths by using arbitrary $In(n_c) - 1$ pairs of nodes. In other words, one of the $In(n_c)$ original subpaths can remain. As shown in Fig.6, the original subpath $SubPath(4,u_2,v_2)$ does not need to be reconstructed. We define the maximum graph G_{MAX} as the secondary graph G_D corresponding to the individual '11…1', where each newly introduced link exists in the graph. Using G_{MAX} is the most likely secondary graph to find alternative paths for reconstruction.

The steps of reconstructing a subpath from u_i to v_i is illustrated in Fig.7. Note that if the original subpath is occupied by multiple paths, update must be made to each of the relevant paths. Take Fig.6(*b*) as an example, the new subpath from u_3 to v_3 is occupied by paths 6 and 8, so the two paths must be updated by using the new subpath.

Fig.8 shows the procedure of the path reconstruction based move (M2). M2 only operates on some subpaths so only the involved subpaths are changed while the remaining structure of $G_{s\rightarrow T}(M2)$ is not affected. It is not difficult to understand that (1) the initial routing subgraph, i.e $G_{s\rightarrow T}(M1)$, influences the performance of M2, and (2) M2 is more effective to avoid coding at a coding node with less incoming links.

After the neighbourhood search, the newly obtained routing subgraph $G_{s\rightarrow T}(M2)$ is returned as the corresponding routing subgraph of the given individual **X**. Instead of $\Phi(G_{s\rightarrow T}(start))$, the fitness value $f(\mathbf{X})$ of **X** is set to $\Phi(G_{s\rightarrow T}(M2))$, i.e. the number of coding links in $G_{s\rightarrow T}(M2)$.

```
1) Delete $G_{s\to T}(M2)$ completely from $G_{MAX}$. Denote the newly obtained subgraph as $G_{NEW}$.
2) Insert the subpath that needs to be reconstructed into $G_{NEW}$.
3) Delete the corresponding incoming link of $n_c$ from $G_{NEW}$.
4) Find a subpath from $u_i$ to $v_i$ from $G_{NEW}$ by using Dijkstra algorithm, a classical shortest path
   algorithm [32]. If it is successful, update $G_{s\to T}(M2)$ by replacing the original subpath in
   $G_{s\to T}(M2)$ with the reconstructed subpath. Otherwise, $G_{s\to T}(M2)$ remains unchanged.
```

Fig.7 Steps for reconstructing a subpath.

```
1) Set $G_{s\to T}(M2) = G_{s\to T}(M1)$.
2) If $G_{s\to T}(M2)$ is coding-free, stop the procedure and output $G_{s\to T}(M2)$.
3) Mark all coding nodes in $G_{s\to T}(M2)$ as untraversed.
4) Randomly select an untraversed coding node in $G_{s\to T}(M2)$, e.g. node $n_c$. Assume $n_c$ has $In(n_c)$ incoming links
   in $G_{s\to T}(M2)$. Number these incoming links in any order. Set counter $c = 0$.
5) For $i = 1$ to $In(n_c)$
    a) Determine the $i$-th pair of start and end points $(u_i,v_i)$. // see steps of determining start and end points.
    b) Find an alternative subpath from $u_i$ to $v_i$. If the subpath is successfully reconstructed, update the relevant
       paths in $G_{s\to T}(M2)$ and set $c = c + 1$. // see Fig.7 for detail.
    d) If $c = In(n_c) - 1$ or $i = In(n_c)$, go to step 6.
6) Mark $n_c$ as a traversed coding node in $G_{s\to T}(M2)$. If all coding nodes in $G_{s\to T}(M2)$ are traversed, stop the
   procedure and output $G_{s\to T}(M2)$; otherwise, go to step 4.
```

Fig.8 Procedure of the path reconstruction based move (M2)

3.4 The Structure of the Proposed Algorithm

The pseudo-code of the proposed EA is shown in Fig.9 with the following two extensions for solving the problem concerned: (1) an OR-based crossover adopted as the recombination operator, and (2) a neighborhood search operator embedded.

Note that the selection and mutation in this paper is the tournament selection and simple mutation, respectively. The tournament selection repeatedly selects the best individual of a randomly chosen subset of the population, where the subset size is referred to as tournament size [12]. In the simple mutation, each bit of an individual is flipped at a given mutation probability.

The genotype encoding approach used in the proposed EA is the binary link state (BLS) encoding. For any outgoing link of an arbitrary merging node with k incoming links, an alphabet of cardinality 2 in the BLS encoding is sufficient to represent all possible 2^k states of k links [11] to the outgoing link.

```
1)  Initialization
2)    Randomly create a population {$\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_{2N}$} with $2N$ individuals
3)    Evaluate each individual $\mathbf{X}_i$, $i = 1,\ldots,2N$, by the following:
4)      if $\mathbf{X}_i$ is feasible then  // see section 3.1 for feasibility verification
5)        Execute the neighborhood search to improve the fitness of $\mathbf{X}_i$ // see section 3.3
6)      else
7)        Assign a sufficiently large value $\Psi$ to $\mathbf{X}_i$ as its fitness value
8)  Repeat
9)    Select a new population from the old population by using tournament selection
10)   Execute the OR-based crossover to each selected pair of individuals at a crossover
      probability $p_c$ // see section 3.2
11)   Execute a simple mutation to the population where each bit is flipped independently at a
      mutation probability $p_m$
12)   Evaluate each individual $\mathbf{X}_i$, $i = 1,\ldots,2N$, by the following:
13)     if $\mathbf{X}_i$ is feasible then  // see section 3.1 for feasibility verification
14)       Execute the neighborhood search to improve the fitness of $\mathbf{X}_i$ // see section 3.3
15)     else
16)       Assign a sufficiently large value $\Psi$ to $\mathbf{X}_i$ as its fitness value
17) until the termination condition is met
```
Fig.9 The pseudo-code of the proposed algorithm

Before initialization, the parameters of the EA, i.e. the population size $2N$, the tournament size, the crossover probability p_c and the mutation probability p_m need to be defined. In the initialization, a population {$\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_{2N}$} is randomly generated and then evaluated. In the evaluation, we check the feasibility of each individual $\mathbf{X}_i$, i.e. verify if the individual corresponds to a secondary graph G_D where a feasible routing subgraph $G_{s\to T}(start)$ can be found. If $\mathbf{X}_i$ is feasible, its corresponding G_D and $G_{s\to T}(start)$ are passed on to the neighborhood search where the two moves, i.e. M1 and M2 are executed to find a new and better routing subgraph from the neighbors of $G_{s\to T}(start)$. After that, the fitness of $\mathbf{X}_i$ is set to the number of coding links in the new routing subgraph, i.e. $\Phi(G_{s\to T}(M2))$. On the other hand, if $\mathbf{X}_i$ is an infeasible individual, we assign a sufficiently large number Ψ to $\mathbf{X}_i$ as its fitness value (in this paper, $\Psi = 50$).

In the loop, the tournament selection selects fitter individuals from the old population to form a new population. Then, the OR-based crossover is performed to each selected pair of individuals to create new and fitter offspring. Later, simple mutation is applied to diversify the population. After mutation, evaluation is performed in the same way as that in the initialization. The selection, OR-based crossover and mutation are repeated iteratively to evolve the population until the evolution stops.

The termination conditions can be: (1) finding a coding-free routing subgraph, or (2) reaching a predefined number of generations.

4. Performance Evaluation

We first investigate the effectiveness of the two extensions in EA, i.e. the OR-based crossover and the neighbourhood search operator, respectively. Experiments were conducted on four testing networks, 3-copies, 7-copies, 15-copies, and 31-copies which are usually used to test the performance of algorithms on the network coding resource minimization problem [11, 24, 26]. The *n*-copies networks are generated based on the network shown in Fig.4(a) by cascading *n*-copies of it, where each sink of the upper copy is a source of the lower copy. The *n*-copies network has $n + 1$ sinks, to which the maximum data transmission rate from the source is 2. The length of each individual is 32 bits in 3-copies network, 80 bits in 7-copies network, 176 bits in 15-copies network, and 368 bits in 31-copies network, respectively. All experimental results are collected by running each algorithm 50 times.

4.1 The effectiveness of the OR-based crossover

As mentioned in section 3.2, the OR-based crossover (ORX) helps to produce feasible individuals during the evolution. We examine the effectiveness of this operator by comparing the performance of the following three algorithms on 3-copies, 7-copies, 15-copies and 31-copies networks:

— GA-UX-M: GA with tournament selection, uniform crossover (UX) and simple mutation. Uniform crossover decides which parent individual will contribute to each position in the offspring individual, with a mixing ratio [12]. In simple mutation, each bit is flipped independently at a mutation probability p_m. GA-UX-M is used to solve the problem concerned in this paper [11].

— GA-M: GA with tournament selection and simple mutation. Crossover is excluded.

— GA-ORX-M: GA with tournament selection, ORX and simple mutation.

As we see, the differences among these algorithms lie in whether they have crossover and if so which crossover they use. For the three algorithms, the initial population is randomly generated with an all-one vector inserted. This is to ensure each algorithm begins with at least one feasible individual [10,11]. The population size in the three experiments is set to 20. The termination condition is 100 generations of evolution. The tournament size for selection is fixed at 2, which is a typical setting for tournament selection [12]. The mutation probability is set to 0.006 [11]. As to the crossover probability p_c, we set 0.25, 0.50, and 0.75 to GA-UX-M and GA-ORX-M.

For each algorithm, the mean and standard deviation (std) of the best solutions obtained over 20 runs are shown in Table 2. First, we can see that GA-ORX-M significantly outperforms GA-UX-M and GA-M in each instance. This evidence demonstrates that the adoption of ORX can effectively improve the performance of GA. Besides, if we look at GA-ORX-M with different crossover probabilities, it is easily seen that $p_c = 0.25$ leads to the best performance. As aforementioned, ORX can increase the number of feasible individuals in the population. A larger p_c is more likely to result into a faster growing of feasible individuals. If p_c is sufficiently large, ORX will restrict the search around a small region (where all-one vector is the centre) in the solution space and harm the population diversity. On the contrary, ORX with a smaller p_c not only slowly compensates for the loss of feasible individuals in the population during the evolution, but also has a lower impact on the diversification of GA. This is why GA-ORX-M with a smaller p_c performs better. Regarding the performance of GA-UX-M and GA-M, one may find that the former only performs slightly better except for the 7-copies network. This evidence shows that using UX may not be appropriate for the GA regarding the optimization problem in this paper.

Table 2 Results of GA-UX-M, GA-M and GA-ORX-M (The Best Results are in Bold)

Algorithm	p_c	3-copies		7-copies		15-copies		31-copies	
		mean	std	mean	std	mean	std	mean	std
GA-UX-M	0.25	0.84	1.36	0.70	1.94	9.48	7.97	25.72	10.02
	0.50	0.30	0.90	0.98	2.35	7.46	6.12	22.48	6.93
	0.75	0.60	1.21	1.04	2.44	6.72	5.83	21.96	8.35
GA-M	/	0.78	1.32	2.28	3.27	8.24	9.68	23.38	8.77
GA-ORX-M	0.25	**0.00**	0.00	**0.00**	0.00	**1.20**	0.92	**11.32**	1.91
	0.50	**0.00**	0.00	**0.00**	0.00	1.78	0.97	13.30	1.86
	0.75	**0.00**	0.00	**0.00**	0.00	3.54	1.77	15.64	2.52

4.2 The effectiveness of the neighbourhood search operator

As discussed in section 3.3, a neighbourhood search operator with two moves, i.e. M1 and M2, is applied to each feasible individual to enhance global exploration. In order to verify the effectiveness of the operator, we carry out the following experiments: collect randomly ten feasible individuals for each of the four testing networks, i.e. 3-copies, 7-copies, 15-copies and 31-copies networks. For each individual, we compare two routing subgraphs obtained before and after implementing the neighbourhood search in terms of the number of coding links, i.e. Φ_{BEF} and Φ_{AFT}. To show the performance of each move, we run neighbourhood search with the following settings: (1) only M1 is included, (2) only M2 is included, and (3) both M1 and M2 are included. For convenience, we number the ten feasible individuals obtained for each network by Φ_{BEF} they have.

The experimental results are shown in Table 3. First, it can be seen that Φ_{AFT} is always smaller than Φ_{BEF} and the difference between them is sometimes significant especially in the 15-copies and 31-copies networks, demonstrating the effect of the neighbourhood search. With respect to the power of each move, we see that (1) M1 and M2 are both feasible and effective. (2) For the 3-copies and 7-copies networks, M1 sometimes performs worse than M2. (3) For the 15-copies and 31-copies networks, M1 is more likely to obtain a routing subgraph containing less number of coding links than M2. As mentioned before, M1 is to delete redundant links from a secondary graph $G_D(M1)$. If $G_D(M1)$ does not have enough links for deletion, the performance of M1 is deteriorated. In addition, the uncertainty of link traversal order also affects the results of M1. On the other hand, M2 only focuses on reconstructing a part of $G_{s\rightarrow T}(M2)$ each time. So, M2 is mainly affected by the input routing subgraph. Last but not least, the results obtained by cascading M1 and M2 also illustrate M2 is a useful complement to M1 as M2 can further optimize the routing subgraph obtained by M1. We therefore conclude that the neighbourhood search operator helps to improve the performance of our algorithm as long as enough feasible individuals are generated.

Table 3 Experimental Results of The Neighbourhood Search

Feasible Individual No.	3-copies			7-copies			15-copies			31-copies		
	Φ_{BEF}	settings	Φ_{AFT}	Φ_{BEF}	settings	Φ_{AFT}	Φ_{BEF}	settings	Φ_{AFT}	Φ_{BEF}	settings	Φ_{AFT}
1	2	M1 only	0	2	M1 only	0	9	M1 only	1	29	M1 only	0
		M2 only	0		M2 only	0		M2 only	0		M2 only	4
		M1&M2	0		M1&M2	0		M1&M2	0		M1&M2	0
2	2	M1 only	0	2	M1 only	0	10	M1 only	1	31	M1 only	0
		M2 only	0		M2 only	0		M2 only	2		M2 only	6
		M1&M2	0		M1&M2	0		M1&M2	0		M1&M2	0
3	2	M1 only	0	3	M1 only	0	10	M1 only	1	32	M1 only	2
		M2 only	0		M2 only	0		M2 only	0		M2 only	3
		M1&M2	0		M1&M2	0		M1&M2	0		M1&M2	0
4	2	M1 only	0	5	M1 only	0	11	M1 only	0	33	M1 only	1
		M2 only	0		M2 only	0		M2 only	0		M2 only	5
		M1&M2	0		M1&M2	0		M1&M2	0		M1&M2	0
5	2	M1 only	0	5	M1 only	0	13	M1 only	2	34	M1 only	0
		M2 only	0		M2 only	0		M2 only	2		M2 only	7
		M1&M2	0		M1&M2	0		M1&M2	0		M1&M2	0
6	2	M1 only	0	6	M1 only	2	14	M1 only	1	35	M1 only	0
		M2 only	0		M2 only	0		M2 only	4		M2 only	5
		M1&M2	0		M1&M2	0		M1&M2	0		M1&M2	0
7	2	M1 only	1	7	M1 only	0	17	M1 only	1	35	M1 only	1
		M2 only	0		M2 only	0		M2 only	4		M2 only	5
		M1&M2	0		M1&M2	0		M1&M2	0		M1&M2	0
8	3	M1 only	0	7	M1 only	1	18	M1 only	0	35	M1 only	1
		M2 only	0		M2 only	0		M2 only	4		M2 only	8
		M1&M2	0		M1&M2	0		M1&M2	0		M1&M2	0
9	3	M1 only	1	7	M1 only	1	19	M1 only	0	35	M1 only	0
		M2 only	0		M2 only	0		M2 only	6		M2 only	7
		M1&M2	0		M1&M2	0		M1&M2	0		M1&M2	0
10	3	M1 only	1	7	M1 only	1	20	M1 only	2	36	M1 only	0
		M2 only	0		M2 only	0		M2 only	6		M2 only	7
		M1&M2	0		M1&M2	0		M1&M2	0		M1&M2	0

4.3 Overall performance evaluation

In order to thoroughly analyse the overall performance of the proposed EA, we compare it with six state-of-the-art algorithms in the literature. The objectives are to minimize coding operations involved while meeting the expected data rate:

− GA1: BLS encoding based GA [11]. Different from GA-UX-M used in section 4.1, GA1 employs a greedy sweep operator after the evolution to improve the quality of the best solution found by flipping each of the remaining 1's to 0 if it results into a feasible solution.

− GA2: GA with the block transmission state encoding and operators [11]. The same greedy sweep operator is applied at the end of evolution as in GA1.

− QEA1: the quantum-inspired evolutionary algorithm (QEA) proposed in [15]. This QEA is featured with a multi-granularity evolution mechanism, an adaptive quantum mutation and a penalty-function-based fitness function.

− QEA2: QEA proposed in [23]. The evolutionary parameters of QEA2 are adaptively adjusted according to the current and previous fitness values of the individual.

− PBIL: Population based incremental learning algorithm [24]. PBIL maintains a real-valued probability vector (PV) which, when sampled, produces promising solutions with higher probabilities. A restart scheme is introduced to help PBIL to escape from local optima.

− cGA: compact genetic algorithm [26]. Similar to PBIL, cGA also maintains a PV. However, PV in cGA is only sampled once at each generation. The new sample is compared with the best-so-far sample and between the two the winner is used to update the PV.

− pEA: path-oriented encoding evolutionary algorithm [22]. Each chromosome is represented by a union of paths originating from the source and terminating at one of the receivers. A local search operator is used to improve the solution quality.

− EA: the proposed algorithm, i.e. GA with ORX, mutation and the neighbourhood search.

The population size for each population-based algorithm is set to 20. The crossover probability p_c and mutation probability p_m are set to 0.8 and 0.006 for GA1 and 0.8 and 0.012 for GA2, respectively [11]. We use the best parameter settings for QEA1, QEA2, PBIL and cGA [15, 23, 24, 26]. In EA, we set $p_c = 0.25$, $p_m = 0.006$. Experiments have been carried out upon four fixed and ten randomly-generated directed networks. Each algorithm terminates when a coding-free routing subgraph (i.e. optimal solution) is found or a predefined generation (200 for each instance) is reached. Table 4 shows the networks and parameter setup. To encourage scientific comparisons, the detailed information of all 14 instances is provided at http://www.cs.nott.ac.uk/~hxx/. Note that the testing networks are relatively small in size and may not be able to precisely reflect the real-world problems.

All experiments were run on a Windows XP computer with Intel(R) Core(TM)2 Duo CPU E8400 3.0GHz, 2G RAM. The results are achieved by running each algorithm 50 times.

Table 4 Experimental Networks

Network	nodes	links	sinks	rate	L
3-copies	25	36	4	2	32
7-copies	57	84	8	2	80
15-copies	121	180	16	2	176
31-copies	249	372	32	2	368
Rand-1	20	37	5	3	43
Rand-2	20	39	5	3	50
Rand-3	30	60	6	3	86
Rand-4	30	69	6	3	112
Rand-5	40	78	9	3	106
Rand-6	40	85	9	4	64
Rand-7	50	101	8	3	145
Rand-8	50	118	10	4	189
Rand-9	60	150	11	5	235
Rand-10	60	156	10	4	297

Note: L: the encoding length of individuals

Table 5 illustrates the mean and standard deviation (std) of the best solutions obtained over 50 runs by the eight algorithms. The mean value of 0.00 indicates an algorithm can obtain an optimal solution (which corresponds to a coding-free routing subgraph) at each run. The results show that EA and pEA can always find an optimal solution for each instance while the others in some instances cannot guarantee an optimal solution. To further support this observation, the statistical results of comparing algorithms by two-tailed *t*-test [31] with 98 degrees of freedom at a 0.05 level of significance are given in Table 6. The result of Algorithm-1↔Algorthm-2 is shown as "+", "–", or "~" when Algorithm-1 is significantly better than, significantly worse than, or statistically equivalent to Algorithm-2, respectively. It can be seen that EA performs no worse but usually better than the other algorithms, indicating positive contributions of the OR-based crossover and neighbourhood search.

On the other hand, pEA also performs well in each instance. This is because the test problems are relatively small in size, i.e. the number of receivers is limited. As aforementioned, pEA maintains a population of path-oriented chromosomes which are two-dimensional. With a small set of receivers, pEA could gain descent performance. However, with the number of receivers growing, the structure of the chromosome becomes increasingly larger and more complex, which would lead to a weakened optimization performance.

Table 5 Experimental Results of Different Algorithms (The Best Results are in Bold)

	GA1		GA2		QEA1		QEA2	
Scenarios	mean	std	mean	std	mean	std	mean	std
3-copies	0.36	0.74	0.08	0.27	**0.00**	0.00	**0.00**	0.00
7-copies	1.96	1.92	0.68	0.84	0.18	0.62	0.48	0.70
15-copies	7.48	5.12	3.66	2.13	3.10	4.18	5.80	1.62
31-copies	28.75	7.97	18.66	22.58	19.10	5.76	20.00	0.00
Rand-1	0.52	0.88	0.44	0.50	**0.00**	0.00	**0.00**	0.00
Rand-2	0.26	0.66	0.02	0.14	**0.00**	0.00	**0.00**	0.00
Rand-3	0.44	0.83	0.02	0.14	**0.00**	0.00	**0.00**	0.00
Rand-4	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rand-5	2.78	2.71	1.16	0.61	0.46	0.50	0.48	0.54
Rand-6	0.22	0.41	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rand-7	1.58	0.92	1.36	0.66	0.66	0.47	0.58	0.53
Rand-8	2.52	1.44	2.28	0.94	0.98	0.82	0.48	0.61
Rand-9	2.82	1.22	2.34	1.34	1.64	0.98	1.94	1.16
Rand-10	3.26	2.68	1.38	0.69	0.66	0.68	0.42	0.64
	PBIL		cGA		pEA		Proposed EA	
Scenarios	mean	std	mean	mean	std	mean	mean	std
3-copies	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
7-copies	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
15-copies	2.14	4.31	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
31-copies	28.90	10.30	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rand-1	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rand-2	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rand-3	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rand-4	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rand-5	0.04	0.28	0.08	0.27	**0.00**	0.00	**0.00**	0.00
Rand-6	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rand-7	0.38	0.60	0.22	0.41	**0.00**	0.00	**0.00**	0.00
Rand-8	0.60	1.56	0.24	0.43	**0.00**	0.00	**0.00**	0.00
Rand-9	0.06	0.23	0.10	0.30	**0.00**	0.00	**0.00**	0.00
Rand-10	**0.00**	0.00	0.08	0.27	**0.00**	0.00	**0.00**	0.00

The average computational time consumed by each algorithm is shown in Table 7. Compared with other algorithms, EA usually spends less computational time. In particular, the saving of time by using EA is sometimes significant, e.g. in 15-copies and 31-copies networks. This is interesting as the neighbourhood search usually needs a considerable amount of time when applied to each feasible individual. In fact, the reason that EA consumes less computational time is, as examined in section 4.2, the neighbourhood search is powerful and makes full use of each feasible individual. So, neighbourhood search is only applied to a small number of feasible individuals before an optimal solution is found. EA stops whenever an optimal solution is found.

To summarize, the proposed EA has similar performance with pEA and outperforms other existing algorithms with respect to the statistical results.

Table 6 The t-Test Results of Comparing Different Algorithms on 14 Instances

Scenarios	EA↔GA1	EA↔GA2	EA↔QEA1	EA↔QEA2	EA↔PBIL	EA↔cGA	EA↔pEA
3-copies	+	+	~	~	~	~	~
7-copies	+	+	+	+	~	~	~
15-copies	+	+	+	+	+	~	~
31-copies	+	+	+	+	+	~	~
Rand-1	+	+	~	~	~	~	~
Rand-2	+	~	~	~	~	~	~
Rand-3	+	~	~	~	~	~	~
Rand-4	~	~	~	~	~	~	~
Rand-5	+	+	+	+	~	+	~
Rand-6	+	~	~	~	~	~	~
Rand-7	+	+	+	+	+	+	~
Rand-8	+	+	+	+	+	+	~
Rand-9	+	+	+	+	~	+	~
Rand-10	+	+	+	+	~	+	~

Table 7 Computational Time Obtained by Different Algorithms (seconds)

Scenarios	GA1	GA2	QEA1	QEA2	PBIL	cGA	pEA	EA
3-copies	0.99	1.61	0.24	0.21	0.10	0.02	0.09	0.02
7-copies	12.42	11.98	8.54	10.41	2.20	0.15	0.33	0.12
15-copies	55.85	49.27	89.88	91.61	66.14	2.09	1.57	1.22
31-copies	232.92	200.73	728.13	750.70	543.64	29.55	20.79	15.31
Rand-1	2.95	3.30	0.73	0.50	0.29	0.23	0.16	0.03
Rand-2	1.14	1.33	0.37	0.40	0.13	0.06	0.11	0.04
Rand-3	5.13	5.07	0.68	0.75	0.23	0.06	0.27	0.11
Rand-4	3.19	3.13	0.57	0.81	0.26	0.12	0.23	0.06
Rand-5	16.57	14.52	13.82	14.38	6.09	5.62	0.63	1.81
Rand-6	3.54	3.34	0.72	0.84	0.17	0.06	0.23	0.05
Rand-7	24.13	20.78	24.35	22.52	24.29	6.83	2.10	11.17
Rand-8	38.37	30.89	38.04	31.47	27.43	20.11	0.95	14.19
Rand-9	62.46	50.73	73.73	73.94	47.29	19.54	1.93	14.11
Rand-10	71.25	55.46	64.12	52.39	31.81	17.42	1.15	9.27

5. Conclusion and Future Work

In this paper, we investigate an improved evolutionary algorithm (EA) to solve the problem of minimizing coding resource in network coding based multicast. Two extensions have been made to improve the performance of our EA. The first extension is an OR-based crossover which performs OR operations to each selected pair of parents. It shows to be able to constantly produce feasible individuals during the evolution. The proposed crossover operator shows to be crucial in designing an effective EA for the highly contained problem concerned. The other extension is a neighborhood search with two moves applied to each feasible secondary graph, leading to an improved routing subgraph. The intensification of search around promising local regions of the search space further improves the effectiveness of the proposed EA. The effectiveness and efficiency of the EA, which is adapted with specific characteristics of the highly constrained problem, have been evaluated via a large amount of systematic simulations. The experimental results show that the proposed EA outperforms other existing algorithms in the literature in terms of high-quality solution and low computational time, which undoubtedly deserves the best algorithm to date. In a word, the proposed EA is a suitable algorithm to solve the problem concerned.

In nature, a telecommunications network is featured with dynamic environment and full of uncertainty. The network environment is changing all the time. In the future, we will extend the coding operations minimization problem by considering the dynamic environment features, such as, the changing topology, nodes/links failures, and dynamic multicast group. In the meanwhile, we will develop more efficient online search algorithms to rapidly trace the optimal (near optimal) solution and respond to the environment change. As in the real world the network scale is relatively large, centralized search algorithms would be inefficient to handle the dynamics and uncertainties in the networks. Hence, we plan to study and develop decentralized online search algorithm(s) which can be implemented in a distributed real world network environment. Besides, to evaluate the performance of the proposed decentralized algorithm, the test problems used in the paper will be extended so as to reflect the dynamic features of the real world networks.

The optimization problem studied in the paper has a single objective, i.e. to minimize the amount of coding operations incurred during the NCM. In this problem, the expected data rate is regarded as a constraint that must be satisfied. Nevertheless, the expected data rate could be considered as an objective as well since it is also interesting to maximize the achievable data rate for the NCM. Hence, in the future, we will extend the original single-objective optimization problem to a bi-objective optimization problem, where the coding cost is minimized and the achievable data rate is maximized simultaneously. The two objectives may conflict with each other, meaning that to improve one objective may weaken the other. There is no such a solution which is optimized in both objectives [33-36]. The optimal solution set for the bi-objective optimization problem is known as the Pareto-optimal set. We plan to investigate the problem by using MOEAs, such as NSGA-II, epsilon-NSGA-II, SPEA2, etc. The outcome of the research could provide an insightful view on how to balance between the coding cost and the achievable data rate for decision makers.

Acknowledgment

This work was supported by NSFC (No. 61171103, No.61372118), Doctoral Fund of MOE (No.20120005130001), the Fundamental Research Funds for the Central Universities (2682014RC23), the China Scholarship Council (CSC), P. R. China, and University of Nottingham, UK.

References

[1] Miller CK (1998) Multicast networking and applications. Pearson Education.

[2] Xu Y, Qu R (2012) A hybrid scatter search meta-heuristic for delay-constrained multicast routing problems. APPL INTELL 36: 229-241.

[3] Jahanshahi M, Dehghan M, Meybodi MR (2013) LAMR: learning automata based multicast routing protocol for multi-channel multi-radio wireless mesh networks. APPL INTELL 38: 58-77.

[4] Ahlswede R, Cai N, Li SYR and Yeung RW (2000) Network information flow. IEEE T INFORM THEORY 46(4):1204-1216.

[5] Li SYR, Yeung RW, and Cai N (2003) Linear network coding. IEEE T INFORM THEORY 49(2):371-381.

[6] Noguchi T, Matsuda T, Yamamoto M (2003) Performance evaluation of new multicast architecture with network coding. IEICE T COMMUN E86-B: 1788-1795.

[7] Wu Y, Chou PA, and Kung SY (2005) Minimum-energy multicast in mobile ad hoc networks using network coding. IEEE T COMMUN 53(11): 1906-1918.

[8] Fragouli C, Boudec JYL, Widmer J (2006) Network coding: an instant primer. COMPUT COMMUN REV 36: 63-68.

[9] Kim M, Ahn CW, Médard M, and Effros M (2006) On minimizing network coding resources: An evolutionary approach. In: Proceedings of Second Workshop on Network Coding, Theory, and Applications (NetCod2006), Boston.

[10] Kim M, Médard M, Aggarwal V, Reilly VO, Kim W, Ahn CW, and Effros M (2007) Evolutionary approaches to minimizing network coding resources. In: Proceedings of 26th IEEE International Conference on Computer Communications (INFOCOM2007), Anchorage, pp 1991-1999.

[11] Kim M, Aggarwal V, Reilly VO, Médard M, and Kim W (2007) Genetic representations for evolutionary optimization of network coding. In: Proceedings of EvoWorkshops 2007, LNCS 4448, Valencia, pp 21-31.

[12] Mitchell M (1996) An introduction to genetic algorithms. MIT Press.

[13] Krasnogor N, Smith JE (2005) A tutorial for competent memetic algorithms: model, taxonomy and design issues. IEEE T EVOLUT COMPUT: 474-488.

[14] Qu R, Burke EK, McCollum B, Merlot LTG, Lee SY (2009) A survey of search methodologies and automated system development for examination timetabling. J SCHEDULING 12: 58-89.

[15] Xing H, Ji Y, Bai L, and Sun Y (2010) An improved quantum-inspired evolutionary algorithm for coding resource optimization based network coding multicast scheme. AEU-INT J ELECTRON C 64(12): 1105-1113.

[16] Langberg M, Sprintson A, and Bruck J (2006) The encoding complexity of network coding. IEEE T INFORM THEORY 52(6): 2386-2397.

[17] Koetter R and Médard M (2003) An algebraic approach to network coding. IEEE ACM T NETWORK 11(5): 782-795.

[18] Ho T, Koetter R, Médard M, Karger DR, Effros M (2003) The benefits of coding over routing in a randomized setting. In: Proceedings of 2003 IEEE International Symposium on Information Theory, Yokohama, Japan, pp.442-442.

[19] Jaggi S, Sanders P, Chou PA, Effros M, Egner S, Jain K, Tolhuizen L. (2005) Polynomial time algorithms for multicast network code construction. IEEE T INFORM THEORY 51: 1973-1982.

[20] Fong SL, Yeung RW (2010) Variable-rate linear network coding. IEEE T INFORM THEORY 56: 2618-2625.

[21] Fragouli, C., Soljanin, E., 2006. Information flow decomposition for network coding. IEEE T INFORM THEORY 52, 829-848.

[22] Xing H, Qu R, Kendall G, and Bai R (2013). A path-oriented encoding evolutionary algorithm for network coding resource minimization. J OPER RES SOC: 1-17. doi:10.1057/jors.2013.79.

[23] Ji Y, Xing H (2011) A memory-storable quantum-inspired evolutionary algorithm for network coding resource minimization. In: Kita E. (Ed.), Evolutionary Algorithm, InTech, pp 363-380.

[24] Xing H, Qu R (2011) A population based incremental learning for network coding resources minimization. IEEE COMMUN LETT 15: 698-700.

[25] Xing H, Qu R (2011) A population based incremental learning for delay constrained network coding resource minimization. In: Proceedings of EvoApplications 2011, Torino, pp 51-60.

[26] Xing H, Qu R (2012) A compact genetic algorithm for the network coding based resource minimization problem. APPL INTELL 36: 809-823.

[27] Ahn CW (2011) Fast and adaptive evolutionary algorithm for minimum-cost multicast with network coding. ELECTRON LETT 47(12): 700-701.

[28] Luong HN, Nguyen HTT and Ahn CW(2012). Entropy-based efficiency enhancement techniques for evolutionary algorithms. INFORM SCIENCES 188: 100–120.

[29] Xing H and Qu R (2013). A nondominated sorting genetic algorithm for bi-objective network coding based multicast routing problems. INFORM SCIENCES 233: 36–53.

[30] Goldberg AV (1985) A new max-flow algorithm. MIT Technical Report MIT/LCS/TM-291, Laboratory for Computer Science.

[31] Walpole RE, Myers RH, Myers SL, Ye K (2007) Probability and statistics for engineers and scientists. Pearson Education.

[32] Dijkstra EW (1959) A note on two problems in connection with graphs. NUMER MATH 1: 267-271.

[33] Kaur M, Kumar A (2013) Method for solving unbalanced fully fuzzy multi-objective solid minimal cost flow problems. APPL INTELL 38: 239-254.

[34] Lwin K, Qu R (2013) A hybrid algorithm for constrained portfolio selection problems. APPL INTELL 39: 251-266.

[35] Zheng YJ, Chen SY (2013) Cooperative particle swarm optimization for multiobjective transportation planning. APPL INTELL 39: 202-216.

[36] Tsakonas A (2014) An analysis of accuracy-diversity trade-off for hybrid combined system with multiobjective predictor selection. APPL INTELL, published online. doi: 10.1007/s10489-013-0507-8.