Journal of the Operational Research Society (2014) 65, 1261–1277

www.palgrave-journals.com/jors/

A path-oriented encoding evolutionary algorithm for network coding resource minimization

Huanlai Xing[1,2]*, Rong Qu[1], Graham Kendall[1,4] and Ruibin Bai[3]

[1]University of Nottingham, Nottingham, UK; [2]Southwest Jiaotong University, Chengdu, China; [3]University of Nottingham Ningbo, Ningbo, China; and [4]University of Nottingham, Malaysia Campus, Kuala Lumpur, Malaysia

Network coding is an emerging telecommunication technique, where any intermediate node is allowed to recombine incoming data if necessary. This technique helps to increase the throughput, however, very likely at the cost of huge amount of computational overhead, due to the packet recombination performed (ie coding operations). Hence, it is of practical importance to reduce coding operations while retaining the benefits that network coding brings to us. In this paper, we propose a novel evolutionary algorithm (EA) to minimize the amount of coding operations involved. Different from the state-of-the-art EAs which all use binary encodings for the problem, our EA is based on path-oriented encoding. In this new encoding scheme, each chromosome is represented by a union of paths originating from the source and terminating at one of the receivers. Employing path-oriented encoding leads to a search space where all solutions are feasible, which fundamentally facilitates more efficient search of EAs. Based on the new encoding, we develop three basic operators, that is, initialization, crossover and mutation. In addition, we design a local search operator to improve the solution quality and hence the performance of our EA. The simulation results demonstrate that our EA significantly outperforms the state-of-the-art algorithms in terms of global exploration and computational time.

Journal of the Operational Research Society (2014) **65**(8), 1261–1277. doi:10.1057/jors.2013.79
Published online 17 July 2013

Keywords: evolutionary computation; multicast routing; network coding

1. Introduction

Network coding is a new routing paradigm, where each intermediate node is not only able to forward the incoming data but also allowed to mathematically recombine (code) them if necessary (Ahlswede *et al*, 2000; Li *et al*, 2003). In essence, by introducing extra computations at intermediate nodes, network coding can efficiently make use of the bandwidth resource of a network and accommodate more information flows than traditional routing (Li *et al*, 2003). Multicast is a routing scheme for one-to-many data transmission, where the same information is delivered from a source to a set of receivers simultaneously (Miller, 1998). When applied in multicast, network coding can always guarantee a theoretically maximal throughput (Ahlswede *et al*, 2000; Li *et al*, 2003). However, performing coding operations will consume extra computational overhead and buffers. Hence, a natural concern is how to route the data from the source to all receivers at the expected data rate while minimizing the number of coding operations necessarily performed.

The above problem is NP-hard (Kim *et al*, 2006) and a number of evolutionary algorithms (EAs) have been proposed (see Section 2.2), where all of them adopt binary encodings to represent chromosomes (see Section 3.2). However, it is observed in this paper that a major weakness of these encodings is that the search space will include a large proportion of infeasible solutions. These solutions are potential barriers during the search and may significantly deteriorate the performance of EAs. This motivates us to investigate a more suitable encoding approach for EAs to effectively address the problem concerned.

In telecommunications, EAs are widely used as an optimizer to select appropriate routes within limited time. When designing EAs, path-oriented encoding is a direct and natural choice since routing itself is to select paths in a network along which the traffic is delivered. In the literature, path-oriented encoding has been adopted by EAs for solving shortest path routing and multicast routing problems. A number of GAs (Ahn and Ramakrishna, 2002; Cheng and Yang, 2010a, b; Yang *et al*, 2010) are employed to find the cost-optimal path connecting the given source and receiver. Each chromosome is represented by a path containing a string of IDs of nodes through which the path passes. Also, EAs are used to construct least-cost spanning trees, where each chromosome is represented by a set of paths from the source to receivers (Palmer and Kershenbaum, 1994; Siregar *et al*, 2005; Oh *et al*, 2006; Cheng and Yang, 2008, 2010b). Similar to constructing a spanning tree, network coding-based multicast (NCM) finds a subgraph which owns multiple paths. Hence, path-oriented encoding could be a potential choice as the chromosome representation to the network coding resource minimization problem.

*Correspondence: Huanlai Xing, School of Computer Science, University of Nottingham, Room C79, Jubilee campus, Wollaton Road, Nottingham, Nottinghamshire NG8 1BB, UK.
E-mail: psxhx@nottingham.ac.uk

However, to our knowledge no research in the literature concerns path-oriented encoding for the problem concerned.

In this paper, we propose an EA using path-oriented encoding to address the network coding resource minimization problem. In this EA, a chromosome is comprised of d basic units (BUs), where d is the number of receivers. Each BU consists of a set of paths connecting the source and a certain receiver, and do not share any common link. The number of paths in each BU is the same, that is, the data rate R. We develop three genetic operators, that is, initialization, crossover and mutation based on the proposed path-oriented encoding. In the initialization, an allelic BU pool is generated for each receiver. Then, each chromosome in the population is created by randomly selecting one BU for each receiver. To explore the search space we use a single-point crossover that operates upon BUs without damaging the structure of any BU. In mutation, a max-flow algorithm is carried out on a BU of a chromosome, chosen based on the mutation probability that is associated with the number of receivers, d. In addition to these genetic operators, we also develop a problem-specific local search operator to improve solution quality and avoid prematurity. Experimental results show that the path-oriented encoding EA is capable of finding optimal solutions in most of the test instances within a very short time, and the proposed EA outperforms the existing EAs due to the new encoding and the well-designed associated operators.

2. Problem formulation and related work

2.1. Problem formulation

In this paper, a communication network is modelled as a directed graph $G=(V, E)$, where V and E are the node set and link set, respectively. Assume each link $e \in E$ is with a unit capacity. Only integer flows are allowed in G; hence, a link is either idle or occupied by a flow of unit rate (Kim *et al*, 2007a, b). An NCM request can be defined as a source $s \in V$ expects to send the same data to a number of receivers $T=\{t_1, \ldots, t_d\} \subset V$ at rate R, where R is an integer (Xing and Qu, 2012, 2013). Each receiver $t \in T$ can receive the data sent from the source at rate R (Kim *et al*, 2007a, b).

Given an NCM request, the task is to find a connected subgraph in G to support the multicast with network coding (Xing and Qu, 2012, 2013). This subgraph is called NCM subgraph (denoted by G_{NCM}). In an NCM subgraph, there are R link-disjoint paths connecting s and each receiver; a *coding node* is a node that performs coding operations; an outgoing link of a coding node is called a *coding link* if the data sent out via this link are a combination of the data received by the coding node. In network G, a non-receiver intermediate node with multiple incoming links is referred to as a *merging node* (Kim *et al*, 2007a, b). Only merging nodes are possible to become coding nodes. The number of coding links is used to estimate the amount of coding operations performed during the data transmission (Langberg *et al*, 2006). More descriptions can be found in Xing and Qu (2012). The following lists some notations.

M_G: the set of merging nodes in G, where $m \in M_G$ is an arbitrary merging node in G

L_m: the set of outgoing links of merging node m, where $e \in L_m$ is an arbitrary outgoing link of node m

σ_e: a binary variable associated with each link $e \in L_m$, $\forall m \in M_G$. $\sigma_e = 1$ if link e is a coding link; $\sigma_e = 0$ otherwise

$\Phi(G_{\mathrm{NCM}})$: the number of coding links in the NCM subgraph

$R(s, t_k)$: the data rate between s and t_k in the NCM subgraph

$p_i(s, t_k)$: the ith link-disjoint path from s to t_k in G_{NCM}, $i = 1, 2, \ldots, R$

The network coding resource minimization problem is defined as to find an NCM subgraph with the minimum number of coding links while satisfying the data rate R, shown as follows:

Minimize:

$$\Phi(G_{\mathrm{NCM}}) = \sum_{\forall m \in M_G} \left(\sum_{\forall e \in L_m} \sigma_e \right) \tag{1}$$

Subject to:

$$R(s, t_k) = R, \quad \forall t_k \in T \tag{2}$$

Objective (1) defines the optimization problem as to minimize the number of coding links; Constraint (2) defines the achievable rate from s and each receiver as exactly R in the NCM subgraph, also indicating that there are R link-disjoint paths between the source and each receiver.

2.2. Related work

By far, a number of EAs have been proposed for solving the minimization problem. These EAs can be classified into four categories, that is, genetic algorithms (GAs), estimation of distribution algorithms (EDAs), EAs with efficiency enhancement techniques, and hybridized EAs.

Kim *et al* developed several GAs to minimize the involved network coding resource. The first GA was only applicable to acyclic networks (Kim *et al*, 2006). Then, a distributed GA was designed for both acyclic and cyclic networks, where a graph decomposition method (see Section 3.1) was proposed to map the target problem to an EA framework (Kim *et al*, 2007a). Besides, two binary encoding approaches, that is, the binary link state (BLS) and the block transmission state (BTS), and their associated operators were evaluated (Kim *et al*, 2007b) (see Section 3.2).

EDAs have also been used to solve the problem. They maintain one or more probability vectors, rather than a population of explicit solutions. The probability vectors, when sampled,

will generate promising solutions with increasingly higher probabilities during the evolution. So far, quantum-inspired evolutionary algorithm (QEAs) and population-based incremental learning algorithm (PBIL) have been developed to optimize the problem concerned (Xing *et al*, 2010; Ji and Xing, 2011; Xing and Qu, 2011a, b).

In addition, Ahn (2011) and Luong *et al* (2012) studied the minimum-cost network coding problem using evolutionary approaches, where entropy-based evaluation relaxation techniques were introduced to EAs in order to reduce the computational cost incurred during the evolution. 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.

Xing and Qu (2012) proposed a hybridized EA, where a local search procedure is designed and incorporated. Strong global exploration and local exploitation capabilities can both be obtained during the evolution.

Note that all the EAs above adopt binary encodings to represent chromosomes. However, these encodings have their intrinsic drawback as the search space may contain many infeasible solutions that would significantly increase the difficulty of tackling the problem. It is hence worth designing a more appropriate encoding scheme for EAs to effectively address the problem.

3. The proposed evolutionary algorithm

We first review the graph decomposition method based on which the path-oriented encoding is designed. We then review the existing encodings for network coding resource minimization, that is, the BLS and the BTS. After that, we describe the new encoding, its associated operators and the overall procedure of the proposed EA.

3.1. The graph decomposition method

The graph decomposition method is a means of explicitly showing how information flows pass through merging nodes in network G. This method decomposes each merging node into a number of auxiliary nodes, as described below (Kim *et al*, 2007a, b).

Suppose merging node i owns $In(i)$ incoming links and $Out(i)$ outgoing links. This node is decomposed into two node sets: $In(i)$ incoming auxiliary nodes and $Out(i)$ outgoing auxiliary nodes. Each incoming link of i is redirected to the corresponding incoming auxiliary node and each outgoing link of i is redirected to the corresponding outgoing auxiliary node. In addition, an auxiliary link is inserted between arbitrary pair of incoming and outgoing auxiliary nodes. Figure 1 shows an example of the graph decomposition. The original graph with source s and receivers t_1 and t_2 is shown in Figure 1(a), where v_1 and v_2 are merging nodes. Figure 1(b) illustrates the decomposed graph, where eight auxiliary links are inserted, showing all possible routes that information flows may pass through v_1 and v_2.

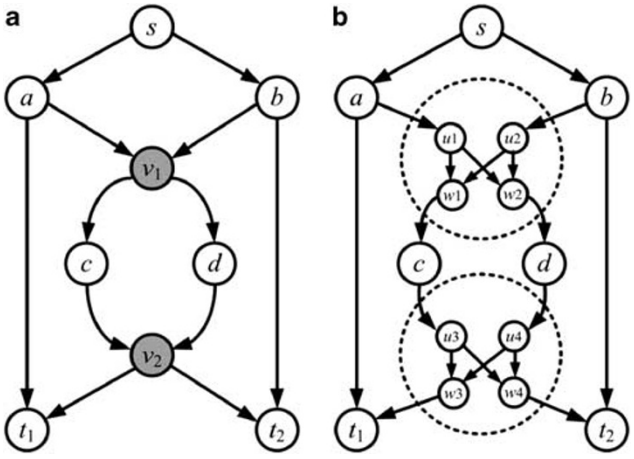

Figure 1 An example of graph decomposition: (a) Original graph; (b) decomposed graph.

3.2. The BLS and BTS encodings

BLS and BTS are the only two existing encoding methods in the literature for the problem concerned (Kim *et al*, 2007a, b). They are based on the graph decomposition method. For an arbitrary merging node with In incoming links and Out outgoing links, there are In auxiliary links heading to each outgoing auxiliary node after graph decomposition, for example, links $u_1 \rightarrow w_1$ and $u_2 \rightarrow w_1$ connect w_1 and links $u_1 \rightarrow w_2$ and $u_2 \rightarrow w_2$ connect w_2, as shown in Figure 1(b). Each auxiliary link can be either active or inactive, indicating whether the link allows flow to pass.

Assume there are OAN outgoing auxiliary nodes in the decomposed graph G_D, where OAN is an integer. In BLS, a chromosome (solution) $\mathbf{X}$ consists of a number of binary arrays b_i, $i = 1, 2, \ldots, OAN$, each determining the states of the auxiliary links heading to a certain outgoing auxiliary node in G_D. In BTS, the chromosome representation is the same as that in the BLS encoding. However, for each array b_i in BTS-based chromosome, once there are at least two 1's in b_i, the remaining 0's in b_i are replaced with 1's.

Using BLS or BTS encoding has two disadvantages. First, the search space contains a considerable amount of infeasible solutions (see Section 4.2). As aforementioned, how flows pass the merging nodes is determined by the states of all incoming auxiliary links in G_D. If many of the incoming auxiliary links are inactive (ie many 0's in chromosome), it is very likely to lead to an infeasible solution. Infeasible solutions are barriers that disconnect feasible regions in the search space and decrease the search efficiency of EAs. Second, the evaluation procedure is complex and indirect, requiring a number of processing steps, that is, chromosome $\mathbf{X} \rightarrow G_D \rightarrow G_{NCM} \rightarrow f(\mathbf{X})$. Meanwhile,

the computational overhead involved in the step $G_D \rightarrow G_{NCM}$ is quite high due to the calculations of the max-flow between the source and each receiver $t_k \in T$. The two drawbacks motivate us to devise a more efficient encoding to represent the solutions to the problem concerned.

3.3. The path-oriented encoding and evaluation

In this paper, we adapt the path-oriented encoding within our proposed EA. Each chromosome consists of a set of paths originating from the source and terminating at one of the receivers. Each path is encoded as a string of positive integers representing the IDs of nodes through which the data pass. The set of paths is grouped into d subsets, that is, d BU, where paths in BU connect to the same receiver and they do not share any common link (ie they are link-disjoint). Besides, there are R paths in each BU, where R is the expected data rate. Each chromosome is feasible since each BU of the chromosome satisfies the data rate requirement. Each BU can be easily obtained by max-flow algorithms. For example, we find a NCM subgraph from Figure 1(b), which consists of four paths, as shown below.

$$p_1(s,t_1) = s \rightarrow a \rightarrow t_1$$

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

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

$$p_2(s,t_2) = s \rightarrow b \rightarrow t_2$$

The corresponding chromosome is illustrated in Figure 2.

Based on the path-oriented encoding, the chromosome evaluation is simple. For chromosome $\mathbf{X}$, the union of all paths in $\mathbf{X}$ forms the corresponding NCM subgraph. The fitness of $\mathbf{X}$, $f(\mathbf{X})$, is known by counting the number of coding links used in the NCM subgraph. So, the computation complexity here is significantly lower than that of BLS and BTS encodings.

Compared with BLS and BTS, path-oriented encoding has two advantages. First, for any instance, the search space consists of feasible solutions only. This leads to a connected search space, and thus helps to reduce the problem difficulty for EAs. Second, the chromosome evaluation is less time-consuming.

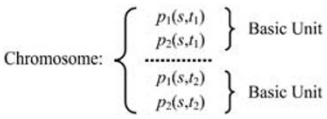

Figure 2 An example chromosome.

3.4. Initialization

It is widely recognized that, for EAs, a good initial population is more likely to lead to a better optimization result. For the proposed algorithm, we initialize the population in the following way. First, we create an allelic BU pool (pool-i) for

```
// Generation of BU pools
for i = 1 to d do
    Set G_temp = G_D and pool-i = ∅
    for j = 1 to pop do
        Find Flow(s,t_i) from G_temp by the max-flow algorithm (Goldberg, 1985)
        if Vol(s,t_i) ≥ R then
            Randomly select R paths from Flow(s,t_i) as a new BU
            if the new BU is not in pool-i then
                Put this BU into pool-i
        Set G_temp = G_D
        Randomly select a BU (with at least one auxiliary link) from pool-i
        Randomly choose an auxiliary link owned by the selected BU
        Delete this auxiliary link from G_temp
// Generation of the population
for j = 1 to pop do
    for i = 1 to d do
        Randomly select a BU from pool-i
        Include the BU in the j-th chromosome
Output the initial population
```

Figure 3 The procedure of initialization.

each receiver t_i, where $i = 1, \ldots, d$. Second, we randomly choose one BU from pool-i, $i = 1, \ldots, d$, and combine the selected BUs as a chromosome. The second step is repeated to create a population of a predefined size.

Let *pop* be the population size and G_D be the decomposed graph. Let R denote the expected data rate and hence each BU contains R link-disjoint paths. Let $Flow(s, t_i)$ and $Vol(s, t_i)$ be the max-flow (made of link-disjoint paths) and its volume from s to receiver t_i, respectively. The max-flow algorithm (Goldberg, 1985) is used to calculate $Flow(s, t_i)$ and $Vol(s, t_i)$. Figure 3 shows the initialization procedure of our EA based on the path-oriented encoding.

For a specific graph G_{temp}, only one BU can be obtained by the max-flow algorithm. To obtain an allelic BU pool for receiver t_i, we have to change the structure of G_{temp} by deleting different links from G_D at each time. As aforementioned, how the information flows pass through a given network depends on the states of all auxiliary links in the decomposed network. So, only the auxiliary links are considered for deletion in our EA. To generate a new BU for receiver t_i, we randomly pick up a BU from pool-i and randomly select an auxiliary link owned by the BU, as shown in steps 9 and 10. The selected link is then removed from G_{temp} to make sure the new G_{temp} is a different graph.

3.5. Crossover

In the proposed EA, we use single-point crossover to each pair of selected chromosomes with a crossover probability p_c. As aforementioned, there are d BUs in a chromosome. The crossover point is randomly chosen from the $(d-1)$ positions between two consecutive BUs. Two offspring are created by swapping the BUs of the two parents after the crossover point. An example crossover operation is illustrated in Figure 4, where each parent consists of four BUs and the crossover point is between the second and third BUs.

First, the proposed crossover does not destroy any BU. So, after crossover, the offspring are all feasible to warrantee a connected search space. No repair is required, which is usually needed in EAs based on the BLS and BTS encodings. Second, the genetic information of the parents could be mixed and spread over offspring chromosomes so that new regions in search space are explored.

3.6. Mutation

Mutation is to help the local exploitation and avoid the prematurity of EAs. As mentioned in Section 3.3, each BU is a set of R link-disjoint paths from the source to a particular receiver. Mutation upon a BU leads to another set of R link-disjoint paths. The idea behind the mutation is that some auxiliary links owned by the chosen BU are deleted from the secondary graph G_D. Then, the new BU is generated by implementing the max-flow algorithm on the new G_D. We propose two mutation operators, the ordinary mutation M_1 and greedy mutation M_2, where each BU of a chromosome is to be

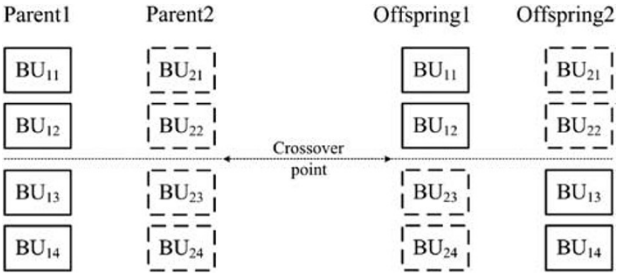

Figure 4 An example of the crossover operator.

mutated with a mutation probability p_m. The difference between M_1 and M_2 is on which links in the chosen BU are deleted. In this paper, we only concern the removal of auxiliary links since they determine the amount of coding resources required.

In M_1, for a chosen BU, we randomly select an auxiliary link in the BU and delete the link from the decomposed graph G_D. After that, we compute the max-flow, that is, $Flow(s, t_i)$, by using the max-flow algorithm on G_D (Goldberg, 1985). If the volume of $Flow(s, t_i)$, $Vol(s, t_i)$, is not smaller than the expected data rate R, a new BU is obtained by randomly selecting R paths in $Flow(s, t_i)$. The new BU then replaces the old BU. If $Vol(s, t_i)$ is smaller than R, the data rate requirement cannot be met and the old BU remains. The procedure of M_1 is shown in Figure 5, where *rnd*() generates a random value uniformly distributed in the range [0,1]. Figure 6 shows an example of BU mutation using M_1, where the example network G and its decomposed network G_D are illustrated in Figure 1. Note that links $u_2 \rightarrow w_1$ and $u_3 \rightarrow w_3$ are the only auxiliary links in the chosen BU. In the example, link $u_3 \rightarrow w_3$ is removed from G_D and a new BU is found based on the new G_D.

In M_1, a random auxiliary link is deleted from G_D to compute a new BU. The new BU, combined with the remaining $(d-1)$ BUs of the chromosome, may lead to an increased number of coding links. This is because no domain knowledge is taken into consideration in M_1. To avoid this we propose the greedy mutation M_2 which is the same as M_1 except the way of which auxiliary links are chosen to be deleted.

In M_2, when deleting auxiliary links from G_D, we concern not only the chosen BU but also the remaining $(d-1)$ BUs. A random auxiliary link owned by the chosen BU is deleted from G_D to make sure that a new different BU is introduced. We also delete in G_D those unoccupied auxiliary links which connect to one of the outgoing auxiliary nodes being occupied by the remaining $(d-1)$ BUs, to make sure that no additional coding links are introduced after M_2. One advantage of M_2 is that the fitness value of a chromosome tends to be smaller after mutation. However, M_2 may lead the search to local optima.

Regarding the mutation probability p_m, a fixed value may not be a wise choice since the number of BUs in a chromosome changes according to d, that is, the number of receivers. A fixed p_m value, for example 0.1, could lead to a dramatically different number of mutation operations during the evolution, which may not be generally applicable for different multicast sessions. In

```
1.  for j = 1 to pop do
2.      for i = 1 to d do
3.          if rnd() < p_m then
                // the i-th BU of the j-th chromosome is chosen
4.              Set G_temp = G_D
5.              if the i-th BU owns at least one auxiliary link then
6.                  Randomly select an auxiliary link owned by the i-th BU
7.                  Delete the link from G_temp
8.                  Compute Flow(s,t_i) from G_temp using the max-flow algorithm
9.                  if Vol(s,t_i) ≥ R then
10.                     Randomly select R paths from Flow(s,t_i) and replace
                        the old BU with the R paths
11. Output the mutated population
```

Figure 5 The procedure of the ordinary mutation M_1.

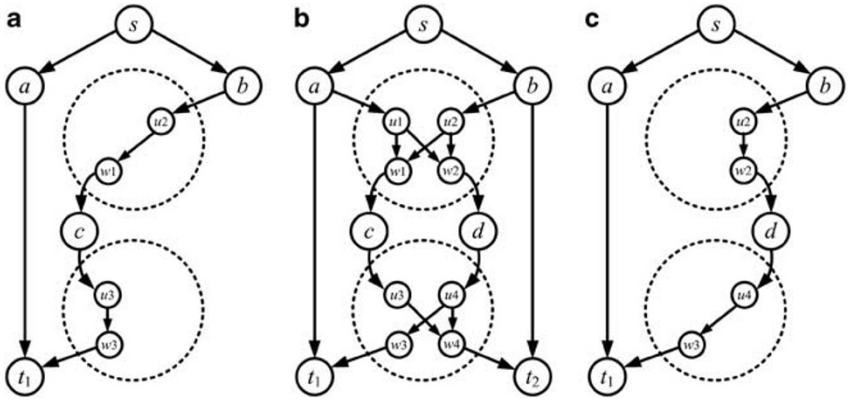

Figure 6 An example of the mutation operator M_1: (a) the chosen BU; (b) link deletion from G_D; (c) the new BU.

our EA, we set $p_m = 1/d$; hence more likely to lead to a stable optimization performance of EA.

3.7. The local search operator

To enhance local exploitation, we propose a local search (LS) operator that is performed on a randomly selected chromosome at each generation.

The aim of this operator is to revise some BUs of the selected chromosome to gradually reduce the number of coding links involved in the multicast. Note that each outgoing link of a merging node is redirected to an outgoing auxiliary node after the graph decomposition, as discussed in Section 3.1. So in the NCM subgraph of an arbitrary chromosome, each coding link corresponds to a certain coding node (ie an outgoing auxiliary node that performs coding). To reduce the number of coding nodes is to decrease the number of coding links. Assume there is a chromosome **X** of which the NCM subgraph contains K coding nodes, where K is an integer. The LS operator aims to remove the occurrence of coding operation at each coding node. The K coding nodes will be processed one by one, in an ascending order according to their node IDs.

We assume the kth coding node (denoted by cnode-k, $k = 1, 2, \ldots, K$) is to be processed by the LS operator. We also assume that there are C ($C \geqslant 2$) auxiliary links connecting to cnode-k in the NCM subgraph of **X,** meaning information via these links is involved in the coding at cnode-k. To remove the coding from cnode-k, one simple idea is to delete arbitrarily $(C-1)$ auxiliary links from the NCM subgraph of **X**. However, directly removing these links leads to an infeasible **X** since BUs which occupy these $(C-1)$ links are damaged. To overcome this, our LS operator reconstructs the affected BUs so that they bypass the use of the $(C-1)$ auxiliary links mentioned above, explained as follows.

First of all, we randomly select $(C-1)$ auxiliary links connecting to cnode-k and check which BUs are occupying

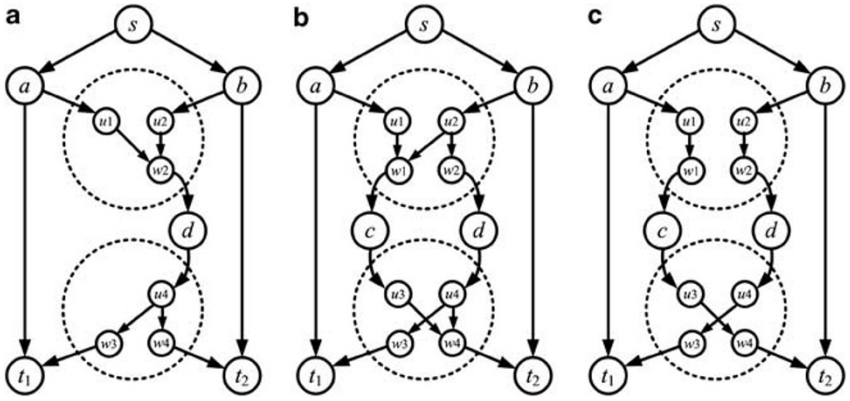

Figure 7 An example of the local search (LS): (a) G_{NCM} before LS; (b) link deletion from G_D; (c) G_{NCM} after LS.

these links. The affected BUs will be reconstructed, while the others remain in the NCM subgraph. Next, we delete the selected $(C-1)$ auxiliary links from the decomposed graph G_D. Besides, we also delete those currently unoccupied auxiliary links from G_D which connect to one of the outgoing auxiliary nodes being occupied by the unaffected BUs. The reason to remove the unoccupied auxiliary links is that we expect to reduce the chance of removing one coding node at the expense of introducing other coding node(s). Finally, we reconstruct the affected BUs by using the max-flow algorithm over G_D. If all the affected BUs are successfully constructed, we obtain a new chromosome $\mathbf{X}_{new}$. If $\mathbf{X}_{new}$ owns less coding links than $\mathbf{X}$, we replace the incumbent $\mathbf{X}$ with $\mathbf{X}_{new}$ (ie the LS moves to an improved solution $\mathbf{X}_{new}$) and repeat the LS operator to improve the new incumbent $\mathbf{X}_{new}$. Otherwise, we retain $\mathbf{X}$ and proceed to the next coding node of $\mathbf{X}$. The LS operator stops when either no improvement is made to the incumbent chromosome after checking all its coding nodes, or a new chromosome with no coding involved (ie optimal) is found.

An example LS is shown in Figure 7, where Figure 1(a) is the example network. The example NCM subgraph G_{NCM} consists of two BUs, that is, $BU_1 = \{s \to a \to t_1,\ s \to b \to u_2 \to w_2 \to d \to u_4 \to w_3 \to t_1\}$ and $BU_2 = \{s \to b \to t_2,\ s \to a \to u_1 \to w_2 \to d \to u_4 \to w_4 \to t_2\}$, as seen in Figure 7(a). Obviously, node w_2 is the only coding node in G_{NCM}. According to the rule of LS, one of the incoming auxiliary links, that is $u_1 \to w_2$ and $u_2 \to w_2$, needs to be removed from G_D. In the example, link $u_1 \to w_2$ is chosen for removal and hence the affected BU, that is BU_2, has to be reconstructed. Besides, as auxiliary nodes w_2 and w_3 are currently occupied by BU_1, all unoccupied auxiliary links heading to w_2 and w_3 also need to be deleted from G_D. So, link $u_3 \to w_3$ is deleted. Based on the new G_D, a new $BU_2 = \{s \to b \to t_2,\ s \to a \to u_1 \to w_1 \to c \to u_3 \to w_4 \to t_2\}$ is rebuilt, as shown in Figure 7 (c). It is easily seen that the combination of BU_1 and BU_2 results into an NCM subgraph without coding operation. Hence, the LS procedure stops and returns the resulting NCM subgraph.

The LS operator is useful to improve the solution-quality (ie better fitness) of the selected chromosome. Also, it changes the structure of the chromosome. Hence, the new chromosome may also help to increase the population diversity. The evaluation of the LS operator is discussed in Section 4.7.

3.8. The overall procedure of the proposed EA

The procedure of the proposed EA is shown in Figure 8. The evaluation of chromosome $X_i(t)$ (in step 4) is simple. In G_D, we mark those nodes and links being occupied by the BUs in $X_i(t)$. The union of the marked nodes and links forms the NCM subgraph G_{NCM} of $X_i(t)$. The number of coding links in G_{NCM}, that is $\Phi(G_{NCM})$, is assigned to $X_i(t)$ as its fitness. In step 8, tournament selection (Mitchell, 1996) is adopted in our proposed EA. The tournament size is set to 2, which is a typical setting for EAs. In step 9, the elitism scheme is used to preserve the best-so-far chromosome. In step 11, either the ordinary mutation or the greedy mutation can be used here. The termination conditions are that, either the EA has found a chromosome of which the NCM subgraph has no coding link, or EA has evolved a predefined number of generations.

4. Performance evaluation

In this section, we first introduce the test instances used to evaluate the performance of the proposed EA (we hereafter call it pEA). We then investigate the deficiency of BLS and BTS encodings. After that we study the effectiveness of the crossover and mutation of pEA, and compare EAs with path-oriented, BLS and BTS encodings. The LS operator is studied next. Finally, we compare pEA with the existing EAs in terms of optimization performance and computational time.

4.1. Test instances

We consider 14 test instances, four on fixed networks and 10 on randomly generated networks. The four fixed networks are 3-copy, 7-copy, 15-copy and 31-copy networks which have

1. **Initialization**
2. Set $t = 0$;
3. Create an initial population $\{\mathbf{X}_1(t), \ldots, \mathbf{X}_{pop}(t)\}$ by using the proposed initialization operator; // see section 3.4
4. Evaluate each chromosome $\mathbf{X}_i(t)$, $i = 1,\ldots, pop$;
5. Randomly select one chromosome and perform LS operator on it; // see section 3.7
6. **Repeat**
7. Set $t = t + 1$;
8. Select a new population $\{\mathbf{X}_1(t), \ldots, \mathbf{X}_{pop}(t)\}$ from the old one by using the tournament selection;
9. Replace a random chromosome with the best chromosome of the previous generation, e.g. $\mathbf{X}_{best}(t\text{-}1)$;
10. Execute crossover to each pair of selected chromosomes with crossover probability p_c; // see section 3.5
11. Execute mutation to each BU of each chromosome with mutation probability p_m; // see section 3.6
12. Evaluate each chromosome $\mathbf{X}_i(t)$, $i = 1,\ldots, pop$;
13. Randomly select one chromosome and perform the LS operator on it; // see section 3.7
14. **until** *the termination condition is met*

Figure 8 The procedure of the proposed EA.

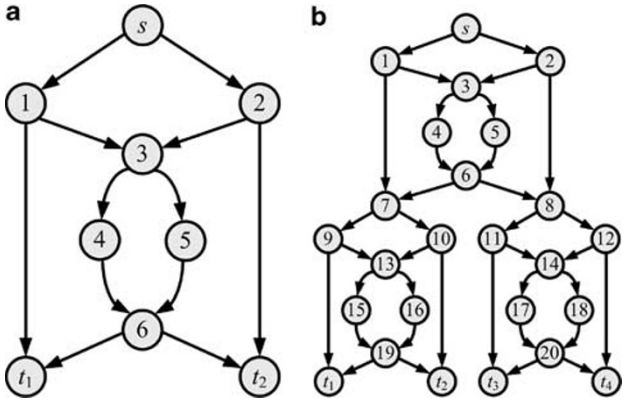

Figure 9 An example of *n*-copy network: (a) original network; (b) 3-copy.

been used to test the performance of EAs for a number of network coding-based optimization problems (Kim *et al*, 2007b; Xing and Qu, 2011a, 2012, 2013). Figure 9 illustrates an example of *n*-copy network, where Figure 9(b) is a 3-copy network constructed by cascading three copies of the original network in Figure 9(a). In an *n*-copy network, the source is the node on the top and the receivers are at the bottom. The *n*-copy network has *n*+1 receivers to which data rate from the source is 2. We hereafter call 3-copy, 7-copy, 15-copy and 31-copy networks as Fix-1, Fix-2, Fix-3 and Fix-4 networks, respectively. The 10 random networks (Rnd-*i*, $i = 1, \ldots, 10$) are directed networks with 20-60 nodes. Table 1 shows the 14 instances and their parameters. To encourage scientific comparisons, all instances are provided at http://www.cs.nott.ac.uk/~rxq/benchmarks.htm. The predefined number of generations for all algorithms tested is set to 200. All experiments were run on a Windows XP computer with Intel(R) Core(TM)2 Duo CPU E8400 3.0 GHz, 2 GB RAM. The results are achieved by running each algorithm 50 times.

Table 1 Experimental networks and instance parameters

Networks	*Original network G*				*Decomposed graph* G_D		
	Nodes	*Links*	*Receivers*	*Rate*	*Nodes*	*Links*	*Auxiliary links*
Fix-1	25	36	4	2	49	68	32
Fix-2	57	84	8	2	117	164	80
Fix-3	121	180	16	2	253	356	176
Fix-4	249	372	32	2	617	740	368
Rnd-1	20	37	5	3	54	81	43
Rnd-2	20	39	5	3	65	89	50
Rnd-3	30	60	6	3	94	146	86
Rnd-4	30	69	6	3	113	181	112
Rnd-5	40	78	9	3	124	184	106
Rnd-6	40	85	9	4	91	149	64
Rnd-7	50	101	8	3	178	246	145
Rnd-8	50	118	10	4	194	307	189
Rnd-9	60	150	11	5	239	385	235
Rnd-10	60	156	10	4	262	453	297

4.2. Deficiency of BLS and BTS encodings

Different encoding approaches could greatly affect the performance of EAs (Mitchell, 1996). The resulting search spaces may be significantly different with respect to not only the size but also the structure and connectivity of the underlying landscape. As discussed in Section 3.2, in theory, the search space of BLS or BTS encoding may contain many infeasible solutions. The solutions are thus scattered in disconnected feasible regions

Table 2 Results of PIS over 10 000 samples (%)

Networks	*BLS*	*BTS*	*Networks*	*BLS*	*BTS*
Fix-1	99.83	99.85	Rnd-4	99.83	99.35
Fix-2	100.00	100.00	Rnd-5	100.00	100.00
Fix-3	100.00	100.00	Rnd-6	99.98	99.91
Fix-4	100.00	100.00	Rnd-7	100.00	100.00
Rnd-1	99.41	99.25	Rnd-8	100.00	100.00
Rnd-2	99.96	99.99	Rnd-9	100.00	100.00
Rnd-3	99.89	99.84	Rnd-10	100.00	100.00

in the search space. The connectivity among feasible solutions may be so weak that to find optimal solution(s) by EAs becomes extremely difficult.

In this section, we statistically measure the proportion of infeasible solutions (PIS) over search space by randomly sampling. Table 2 shows the results of PIS over 10 000 samples for each instance. For all instances, the PIS values are more than 99%. In particular, in Fix-2,3,4 and Rand-5,7,8,9,10, the PISs of BLS and BTS are always 100%, meaning that all samples are infeasible solutions that constitute the majority of the search space. Large amount of infeasible solutions could disconnect feasible regions in the search space and dramatically increase the problem difficulty for search algorithms. Hence, the BLS and BTS encodings are not appropriate encoding schemes for our target problem.

4.3. Performance measures

To show the performance of pEA in various aspects, such as the optimal solution obtained, the convergence characteristic, and the consumed running time, the following performance metrics are used throughout Section 4.

- Mean and standard deviation (SD) of the best solutions found over 50 runs. One best solution is obtained in one run. The mean and SD are important metrics to show the overall performance of a search algorithm.
- Student's *t*-test (Yang and Yao, 2005; Walpole *et al*, 2007) to compare two algorithms (eg A1 and A2) in terms of the fitness values of the 50 best solutions obtained. In this paper, two-tailed *t*-test with 98 degrees of freedom at a 0.05 level of significance is used. The *t*-test result can show statistically if the performance of A1 is better than, worse than, or equivalent to that of A2.
- Successful ratio (SR) of finding an optimal solution in 50 runs. The successful ratio, to some extent, reflects the global exploration ability of an EA to find optimal solutions.
- Evolution of the best fitness averaged over 50 runs. The plot of the evolution illustrates the convergence process of an algorithm.
- Average computational time (ACT) consumed by an algorithm over 50 runs. This metric is a direct indication of the time complexity of an algorithm.

Table 3 Comparisons of pEA with different crossover probabilities Best results are in bold

Networks	$p_c = 0.0$		$p_c = 0.3$		$p_c = 0.6$		$p_c = 0.9$	
	Mean	*SD*	*Mean*	*SD*	*Mean*	*SD*	*Mean*	*SD*
Fix-1	2.84	0.37	1.70	0.61	1.32	0.51	**1.08**	0.27
Fix-2	9.58	1.45	7.64	1.43	6.78	1.35	**6.16**	1.23
Fix-3	22.88	0.47	20.68	1.92	19.74	2.00	**17.54**	1.98
Fix-4	46.94	0.42	45.32	1.89	44.72	1.79	**43.20**	2.26
Rnd-1	2.44	0.64	1.70	0.64	1.18	0.66	**0.96**	0.66
Rnd-2	0.62	0.56	0.12	0.32	0.04	0.19	**0.02**	0.14
Rnd-3	2.64	0.56	1.86	0.70	1.40	0.72	**1.22**	0.64
Rnd-4	0.72	0.45	0.38	0.49	0.22	0.41	**0.10**	0.30
Rnd-5	7.58	0.81	5.60	1.08	5.06	1.13	**4.46**	1.32
Rnd-6	0.40	0.49	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rnd-7	3.86	1.01	3.06	0.79	2.96	1.02	**2.34**	0.77
Rnd-8	6.84	0.42	5.76	1.04	5.28	1.10	**4.64**	1.10
Rnd-9	6.00	0.00	5.42	0.67	5.14	0.63	**4.98**	0.58
Rnd-10	7.94	1.39	6.40	1.22	5.54	1.51	**5.18**	1.17

4.4. The effectiveness of crossover in pEA

As mentioned in Section 3.5, the single-point crossover is used in pEA. We investigate the feasibility of this operator and the impact of different settings of the crossover probability p_c on the performance of pEA. Mutation and LS operator is excluded in pEA in this experiment. We set the population size $pop = 20$ and compare the performance of pEA with four different p_c, that is 0.0, 0.3, 0.6 and 0.9, where $p_c = 0.0$ means the algorithm stops after initialization since no crossover is involved. By comparing the results of different p_c and those of $p_c = 0.0$, one could see the effectiveness of the crossover.

The results of the mean and standard deviation of pEA with different p_c are shown in Table 3. It can be seen that pEA with crossover performs better than pEA without crossover in each instance, indicating crossover can properly drive the evolution process. Besides, we find with larger p_c the mean and SD become increasingly better. The variant of pEA with $p_c = 0.9$ performs the best, showing that rapid exchange of genetic information over different chromosomes helps to explore different areas in the search space. However, we may also find that there remain big gaps between the best solutions obtained by pEA with only crossover and the optimal solutions in each instance. This is mainly because employing crossover only is not enough to guide pEA to escape from local optima. We need mutation to enhance local exploitation and avoid prematurity.

4.5. The effectiveness of mutation in pEA

We propose two mutation operators with $p_m = 1/d$ in Section 3.6, that is, the ordinary mutation M_1 and greedy mutation M_2, where d is the number of receivers. To mutate a BU, M_1 deletes a random auxiliary link of the BU from G_D while M_2 deletes a random auxiliary link of the BU and a number of unoccupied auxiliary links from G_D. The removal of the random link is to

Table 4 Results of mean and standard deviation for different mutations and different mutation probabilities

Networks	$M_1(2/d)$		$M_1(1/d)$		$M_1(0.5/d)$		$M_2(2/d)$		$M_2(1/d)$		$M_2(0.5/d)$	
	Mean	*SD*	*Mean*	*SD*	*Mean*	*SD*	*Mean*	*SD*	*Mean*	*SD*	*Mean*	*SD*
Fix-1	1.00	0.00	1.00	0.00	1.00	0.00	**0.04**	0.19	0.14	0.35	0.26	0.44
Fix-2	4.06	0.23	4.00	0.00	4.00	0.00	**1.26**	0.59	1.64	0.80	1.94	0.79
Fix-3	10.52	1.05	8.50	0.54	8.44	0.57	**5.72**	1.22	6.04	0.92	6.68	0.84
Fix-4	29.08	1.81	24.30	0.92	23.54	0.88	**17.60**	1.50	18.20	1.19	18.30	1.55
Rnd-1	**0.00**	0.00	**0.00**	0.00	0.06	0.23	**0.00**	0.00	0.04	0.19	0.06	0.23
Rnd-2	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rnd-3	0.02	0.14	**0.00**	0.00	0.06	0.23	**0.00**	0.00	0.04	0.19	0.02	0.14
Rnd-4	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rnd-5	1.88	0.43	1.40	0.53	1.50	0.61	**0.00**	0.00	0.02	0.14	0.12	0.32
Rnd-6	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rnd-7	1.06	0.23	1.02	0.14	1.10	0.30	**0.12**	0.32	0.34	0.47	0.56	0.50
Rnd-8	2.10	0.30	2.16	0.37	2.34	0.51	**0.02**	0.14	0.04	0.19	0.30	0.46
Rnd-9	2.46	0.57	2.06	0.46	2.36	0.66	**0.80**	0.40	0.86	0.35	0.94	0.23
Rnd-10	1.92	0.48	1.62	0.49	1.76	0.59	**0.00**	0.00	0.00	0.00	0.06	0.23

Best results are in bold

make sure that the mutated BU is different from the old one. Besides, the removal of those unoccupied links is to ensure no extra coding link will be introduced after mutation.

In the following experiment, we compare the performance of pEA with the proposed crossover and different mutations. The comparison between M_1 and M_2 can show whether the removal of those unoccupied auxiliary links helps to improve the performance of pEA. When comparing M_1 and M_2, we also study the impact of different p_m, that is $2/d$, $1/d$ and $0.5/d$. Let $M_1(p_m)$ and $M_2(p_m)$ denote the two mutations with p_m, respectively. In the experiment, LS operator is excluded. We set $pop = 20$ and $p_c = 0.9$.

Table 4 shows the results of mean and standard deviation of the obtained best fitness values by pEA with different mutations and different p_m. Between the two mutations, we find that pEA with M_2 performs better than pEA with M_1 if taking into account the results for all instances. The worst p_m for M_2 is $0.5/d$ while the best p_m for M_1 is $1/d$. If comparing the results of $M_2(0.5/d)$ and those of $M_1(1/d)$, we see that $M_2(0.5/d)$ wins in nine instances while $M_1(1/d)$ wins in two instances, indicating M_2 is more effective than M_1. In addition, having a look at M_2 with different p_m, we also find that the mean and SD become better and better with p_m changing from $0.5/d$ to $2/d$. This is because when mutating a BU, M_2 makes sure that the rebuilt BU does not increase the amount of coding operations to the corresponding chromosome. On the contrary, it is possible that coding at one or more nodes of a chromosome is eliminated after M_2. Hence, imposing reasonably more M_2 operations to the evolving population is more likely to obtain a better optimization performance of pEA. We hereafter only use the greedy mutation as the mutation operator in our pEA.

To further support our findings, we compare different mutations with different p_m by using Student's t-test (see Section 4.3), where results are given in Table 5. The result of comparison between A1↔A2 is shown as '+', '−', or '∼' when A1 is significantly better than, significantly worse than, or statistically equivalent to A2, respectively. The table shows that M_2 is significantly better than M_1 in nine instances and statistically equivalent to M_1 in the remaining instances, which undoubtedly reflects the superiority of M_2 over M_1. Moreover, M_2 with a larger p_m performs better than M_2 with a smaller p_m. However, their performances do not differ too much. For example, between $M_2(2/d)$ and $M_2(1/d)$, the former only wins two instances.

The results of the successful ratio and average computational time are collected in Table 6. For the successful ratio, the results match our findings from Table 4, where M_2 is better than M_1 and a larger p_m results into a better performance of M_2. For the average computational time, we find that the computational complexity of mutation is higher than that of evaluation.

In general, fitness evaluation is assumed to be the most time-consuming operation compared with other operations such as selection, crossover and mutation for highly complex optimization problems. However, the above assumption is no longer held in pEA (without the LS operator) where mutation takes a comparable larger computation time over the fitness evaluation. In mutations (ie M_1 and M_2), computation is spent on two steps, that is, the removal of some auxiliary links from the decomposed graph G_D and the reconstruction of a new BU. The max-flow algorithm in Goldberg (1985) is used, leading to a time complexity of $O(|V_D|^2 \cdot |E_D|^{1/2})$, where $|V_D|$ and $|E_D|$ are the number of nodes and links in G_D, respectively. Compared with the reconstruction of the BU, the removal of auxiliary links consumes very limited computation and can be ignored. Hence, to mutate a chromosome (no matter M_1 or M_2), we require a complexity of O_M, where $O_M = O(p_m \cdot d \cdot |V_D|^2 \cdot |E_D|^{1/2})$. In contrast, to evaluate a chromosome, we only need to obtain the NCM subgraph G_{NCM} of this chromosome and check how many outgoing auxiliary nodes perform coding in G_{NCM}. As mentioned in Section 3.3, each G_{NCM} consists of d BUs, each of which contains R paths, for example $p_i(s, t_k)$ is the ith path of the kth BU. Let L_{ik} be the

Table 5 *t*-test results for different mutations and different mutation probabilities

Networks	$M_2(2/d)\leftrightarrow M_2(1/d)$	$M_2(2/d)\leftrightarrow M_2(0.5/d)$	$M_2(1/d)\leftrightarrow M_2(0.5/d)$	$M_2(2/d)\leftrightarrow M_1(2/d)$	$M_2(2/d)\leftrightarrow M_1(1/d)$	$M_2(2/d)\leftrightarrow M_1(0.5/d)$
Fix-1	~	+	~	+	+	+
Fix-2	+	+	~	+	+	+
Fix-3	~	+	+	+	+	+
Fix-4	~	~	~	+	+	+
Rnd-1	~	~	~	~	~	~
Rnd-2	~	~	~	~	~	~
Rnd-3	~	~	~	~	~	~
Rnd-4	~	~	~	~	~	~
Rnd-5	~	+	~	+	+	+
Rnd-6	~	~	~	~	~	~
Rnd-7	+	+	+	+	+	+
Rnd-8	~	+	+	+	+	+
Rnd-9	~	+	~	+	+	+
Rnd-10	~	~	~	+	+	+

	$M_2(1/d)\leftrightarrow M_1(2/d)$	$M_2(1/d)\leftrightarrow M_1(1/d)$	$M_2(1/d)\leftrightarrow M_1(0.5/d)$	$M_2(0.5/d)\leftrightarrow M_1(2/d)$	$M_2(0.5/d)\leftrightarrow M_1(1/d)$	$M_2(0.5/d)\leftrightarrow M_1(0.5/d)$
Fix-1	+	+	+	+	+	+
Fix-2	+	+	+	+	+	+
Fix-3	+	+	+	+	+	+
Fix-4	+	+	+	+	+	+
Rnd-1	~	~	~	~	~	~
Rnd-2	~	~	~	~	~	~
Rnd-3	~	~	~	~	~	~
Rnd-4	~	~	~	~	~	~
Rnd-5	+	+	+	+	+	+
Rnd-6	~	~	~	~	~	~
Rnd-7	+	+	+	+	+	+
Rnd-8	+	+	+	+	+	+
Rnd-9	+	+	+	+	+	+
Rnd-10	+	+	+	+	+	+

Table 6 Results of successful ratio and average computational time

Networks	*SR (%)*						*ACT (sec.)*					
	$M_1(2/d)$	$M_1(1/d)$	$M_1(0.5/d)$	$M_2(2/d)$	$M_2(1/d)$	$M_2(0.5/d)$	$M_1(2/d)$	$M_1(1/d)$	$M_1(0.5/d)$	$M_2(2/d)$	$M_2(1/d)$	$M_2(0.5/d)$
Fix-1	0	0	0	96	86	74	8.89	4.7475	2.6437	0.39	0.61	0.69
Fix-2	0	0	0	6	8	2	23.97	12.50	7.29	22.29	11.72	7.81
Fix-3	0	0	0	0	0	0	67.80	40.88	22.81	80.69	38.52	22.51
Fix-4	0	0	0	0	0	0	253.66	153.67	88.27	306.12	180.47	100.42
Rnd-1	100	100	94	100	96	94	2.21	0.69	0.61	0.20	0.39	0.33
Rnd-2	100	100	100	100	100	100	0.13	0.10	0.09	0.11	0.09	0.09
Rnd-3	98	100	94	100	96	98	5.12	1.55	1.80	0.42	0.77	0.49
Rnd-4	100	100	100	100	100	100	0.39	0.29	0.23	0.25	0.18	0.20
Rnd-5	0	0	0	100	98	88	30.84	17.23	9.03	4.07	2.85	3.66
Rnd-6	100	100	100	100	100	100	0.23	0.22	0.21	0.23	0.20	0.21
Rnd-7	0	0	0	88	66	44	38.36	20.65	13.23	11.22	11.31	9.49
Rnd-8	0	0	0	98	96	70	59.22	30.28	17.76	10.61	10.00	12.97
Rnd-9	0	0	0	20	14	6	76.46	47.11	28.57	74.10	38.86	29.85
Rnd-10	0	0	0	100	100	94	108.34	57.63	32.65	6.81	9.01	11.43

string length of $p_i(s, t_k)$ in the chromosome. To obtain a G_{NCM} from the corresponding chromosome, the amount of computation involved is $\sum_i \sum_k L_{ik}$, where $L_{ik}<|V_D|$. We assume there are Y outgoing auxiliary nodes in G_D where $Y<|V_D|$ since at least the source and receivers are not decomposed. To check the status of all outgoing auxiliary nodes in G_{NCM}, the amount of computation involved is Y. Therefore, to evaluate a chromosome requires a complexity of $O_E<O(|V_D|^2)<O_M$.

Table 7 Comparisons of GA with different encoding approaches

Networks	*Mean and SD*						*SR (%)*			*ACT (sec.)*		
	BLSGA		*BTSGA*		*pEA*		*BLSGA*	*BTSGA*	*pEA*	*BLSGA*	*BTSGA*	*pEA*
	Mean	*SD*	*Mean*	*SD*	*Mean*	*SD*						
Fix-1	0.46	1.01	0.74	1.20	**0.14**	0.35	80	68	86	1.13	1.47	0.61
Fix-2	3.82	4.26	3.86	3.93	**1.64**	0.80	8	2	8	11.47	11.85	10.72
Fix-3	7.92	5.64	11.92	6.00	**6.04**	0.92	0	0	0	54.57	51.19	38.52
Fix-4	37.60	9.19	43.22	4.47	**18.20**	1.19	0	0	0	98.51	72.55	180.47
Rnd-1	0.96	1.29	1.00	1.48	**0.04**	0.19	46	54	96	3.17	2.86	0.39
Rnd-2	0.44	0.83	0.38	0.75	**0.00**	0.00	78	78	100	0.91	1.12	0.09
Rnd-3	0.40	0.98	0.66	1.20	**0.04**	0.19	84	74	96	4.02	4.21	0.77
Rnd-4	0.28	0.45	0.08	0.27	**0.00**	0.00	72	92	100	2.95	1.98	0.18
Rnd-5	2.98	4.01	4.22	4.70	**0.02**	0.14	8	10	98	15.75	13.45	2.85
Rnd-6	0.42	0.81	0.36	0.77	**0.00**	0.00	78	82	100	3.05	2.67	0.20
Rnd-7	2.14	1.95	2.72	2.16	**0.34**	0.47	10	6	66	21.11	19.15	11.31
Rnd-8	3.04	1.94	3.88	1.96	**0.04**	0.19	2	0	96	32.60	29.23	10.00
Rnd-9	3.68	1.40	4.24	1.59	**0.86**	0.35	0	2	14	51.49	45.81	38.86
Rnd-10	3.52	3.40	3.76	3.50	**0.00**	0.00	4	0	100	62.04	57.25	9.01

Note: Best results are shown in bold.

According to the above finding, the computational time in pEA is mainly spent on the mutation operations during the evolution. Hence, the computational time of pEA should be proportional to the amount of mutation operations. Let us take some examples to show the linear relationship between them. Note that pEA stops at two conditions, either a chromosome without coding is found or a predefined number of generations is reached. To show if the computational time changes proportionally to the amount of mutation operations during the evaluation, we should look at those instances where the successful ratios for different mutation rates are all 0%. In these instances the amount of mutation operations for different p_m is proportional and we only need to check if the computational time is also proportional. Taking instance Fix-3 as an example, theoretically, the ratio of the amount of mutations during the evolution for $M_2(2/d)$, $M_2(1/d)$ and $M_2(0.5/d)$ is 4:2:1. In practice, the ratio of the average computational time of $M_2(2/d)$, $M_2(1/d)$ and $M_2(0.5/d)$ are calculated as 3.58:1.71:1.00 which is similar to the theoretical ratio.

4.6. Comparisons of different encoding approaches

In this section, we show the superiority of the path-oriented encoding over other existing encoding approaches by comparing the performance of three EAs, that is pEA, GA with BLS encoding (BLSGA) and GA with BTS encoding (BTSGA). For the BLS and BTS encoding approaches, please see Kim *et al* (2007b) and Section 3.2 for details. Note that an all-one chromosome is inserted into the initial population of BLSGA and BTSGA to make sure they begin with at least one feasible solution; otherwise, the two GAs may never converge since no feasible solution may be obtained during the search (Kim *et al*, 2007b). This has showed to be an effective method in previous work (Kim *et al*, 2007a, b; Xing and Qu, 2011a, b, 2012, 2013).

Table 8 *t*-test results for different GAs

Networks	*pEA↔ BLSGA*	*pEA↔ BTSGA*	*Networks*	*pEA↔ BLSGA*	*pEA↔ BTSGA*
Fix-1	+	+	Rnd-4	+	+
Fix-2	+	+	Rnd-5	+	+
Fix-3	+	+	Rnd-6	+	+
Fix-4	+	+	Rnd-7	+	+
Rnd-1	+	+	Rnd-8	+	+
Rnd-2	+	+	Rnd-9	+	+
Rnd-3	+	+	Rnd-10	+	+

The comparison is based on a standard GA framework, where genetic operators in each EA include selection, crossover and mutation. The population size and the tournament size are set to 20 and 2 for each algorithm, respectively. In pEA, we use the greedy mutation and set $p_c = 0.9$ and $p_m = 1/d$. We adopt the best parameter settings for BLSGA and BTSGA in Kim *et al* (2007b). In BLSGA, $p_c = 0.8$ and $p_m = 0.006$. In BTSGA, $p_c = 0.8$ and $p_m = 0.012$. Besides, BLSGA and BTSGA use the uniform crossover with a mixing ratio of 0.5 and a simple mutation where each bit of a chromosome is flipped at p_m.

The performance comparisons of EAs with different encodings are shown in Table 7. Besides, the *t*-test results are provided in Table 8. Undoubtedly, pEA achieves better optimization results and consumes less ACT than BLSGA and BTSGA in almost all instances.

To show the convergence of the three EAs, we plot the evolution of the best fitness in each generation, averaged over 50 runs for two fixed and four random instances, as shown in Figure 10. First, we can see that pEA always obtains better initial solutions than BLSGA and BTSGA. For example, in Figure 10(a), at the beginning of the evolution, the average best fitness for pEA is around 7 while those of BLSGA and

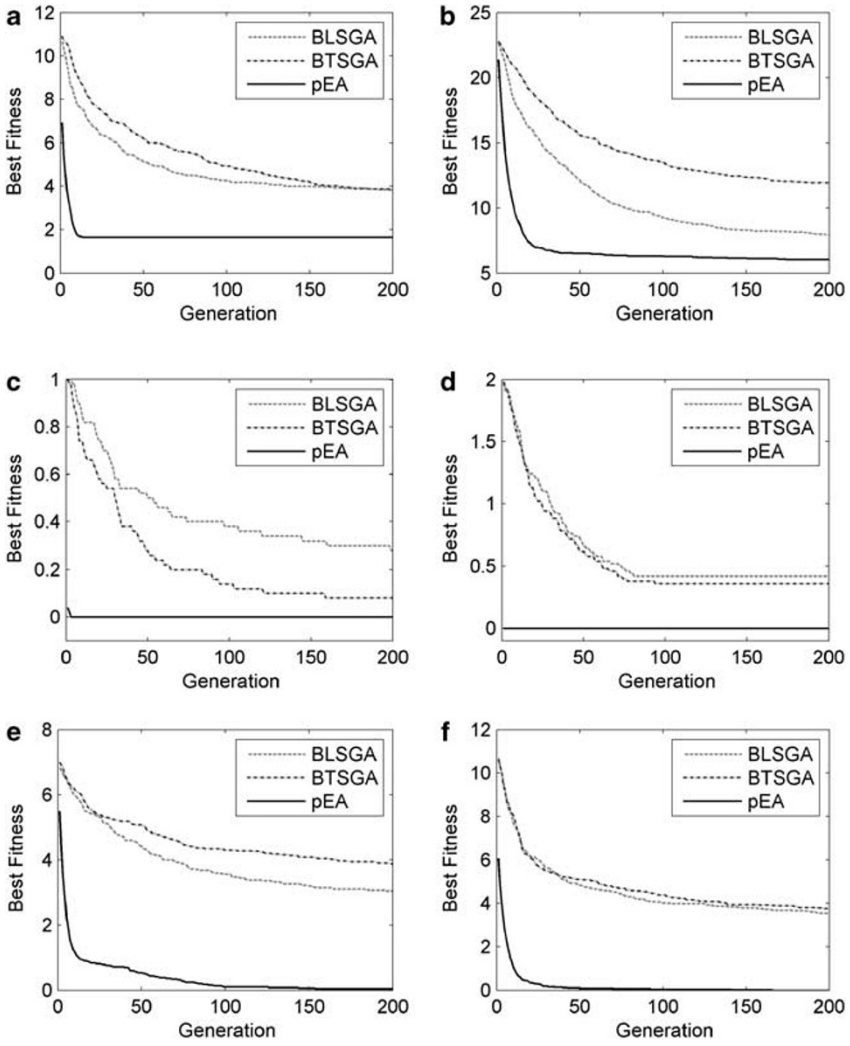

Figure 10 Best fitness *versus* generation for six instances: (a) Fix-2; (b) Fix-3; (c) Rnd-4; (d) Rnd-6; (e) Rnd-8; (f) Rnd-10.

BTSGA are both 11. Moreover, we find that pEA converges very fast especially in the early generations. To find a good solution, pEA needs much less generations than BLSGA and BTSGA. This is an outstanding advantage of pEA especially in real-time and dynamic applications, where a decent solution must be found within a very short time.

Based on the analysis above, we conclude that the path-oriented encoding is more efficient than the BLS and BTS encodings in terms of global optimization, convergence and computational time.

4.7. The effectiveness of the LS operator

As discussed in Section 3.7, a LS operator is applied to a randomly chosen chromosome at each generation to improve the solution quality. To verify the effectiveness of this operator, we randomly construct five chromosomes for each instance by using the initialization method in Section 3.4. We apply the LS operator on each chromosome and compare the fitness values of the chromosome before and after implementing the LS operator, that is Φ_{BEF} and Φ_{AFT}. Let $\mathbf{X}$ and $\mathbf{X}'$ denote the chromosome before and after the LS, and $E_{\mathrm{A}}(\mathbf{X})$ and $E_{\mathrm{A}}(\mathbf{X}')$ be the set of auxiliary links owned by $\mathbf{X}$ and $\mathbf{X}'$, respectively. We define the structural difference coefficient (SDC) ρ between $\mathbf{X}$ and $\mathbf{X}'$ according to the Marczewski-Steinhaus concept of distance (Marczewski and Steinhaus, 1958), as follows:

$$\rho = \frac{\mid E_{\mathrm{A}}(\mathbf{X}) \cup E_{\mathrm{A}}(\mathbf{X}') \mid - \mid E_{\mathrm{A}}(\mathbf{X}) \cap E_{\mathrm{A}}(\mathbf{X}') \mid}{\mid E_{\mathrm{A}}(\mathbf{X}) \cup E_{\mathrm{A}}(\mathbf{X}') \mid} \cdot 100\,\% \quad (3)$$

The value of SDC is between 0.0 and 1.0, which tells us to what degree $\mathbf{X}$ and $\mathbf{X}'$ are different, showing the effect of LS operator on the structure change of solutions. A larger SDC

Table 9 Results of the LS operator

Networks	*Solution* 1			*Solution* 2			*Solution* 3			*Solution* 4			*Solution* 5		
	Φ_{BEF}	Φ_{AFT}	ρ (%)	Φ_{BEF}	Φ_{AFT}	ρ (%)	Φ_{BEF}	Φ_{AFT}	ρ (%)	Φ_{BEF}	Φ_{AFT}	ρ (%)	Φ_{BEF}	Φ_{AFT}	ρ (%)
Fix-1	3	0	54.5	4	0	20.0	3	0	30.0	5	0	36.3	6	0	50.0
Fix-2	12	0	51.7	18	0	53.1	13	0	54.8	16	0	56.2	14	0	53.3
Fix-3	30	0	54.5	35	0	56.3	27	0	54.5	29	0	55.2	40	0	52.8
Fix-4	47	0	53.4	69	0	53.8	60	0	53.9	71	0	54.9	70	0	57.3
Rnd-1	4	2	36.0	5	3	17.3	2	0	30.0	7	3	26.9	6	1	38.4
Rnd-2	2	0	8.33	3	2	18.5	5	3	7.41	4	2	11.5	3	1	33.3
Rnd-3	3	0	38.8	3	0	19.3	5	1	34.1	7	0	45.2	6	0	50.0
Rnd-4	4	1	25.7	5	2	25.6	4	0	22.8	1	0	13.7	2	1	42.1
Rnd-5	12	7	20.0	9	3	17.8	11	3	21.8	8	4	23.6	10	2	39.6
Rnd-6	2	0	21.7	1	0	14.2	1	0	44.0	3	0	24.0	2	0	25.0
Rnd-7	8	4	20.0	6	2	28.3	4	1	10.2	9	3	13.5	6	5	8.33
Rnd-8	8	5	10.5	12	7	15.8	11	3	29.8	15	8	20.4	14	5	22.0
Rnd-9	13	7	15.7	18	5	21.7	8	4	12.3	14	4	19.7	12	7	15.9
Rnd-10	12	3	30.8	15	5	24.5	8	5	8.89	9	5	10.9	7	3	31.1

indicates a severer structural change caused by the LS operator.

The experimental results of Φ_{BEF}, Φ_{AFT} and ρ are shown in Table 9. First, it is seen that Φ_{END} is smaller than Φ_{START} especially for instances Fix-3,4, showing that the LS operator can improve the quality of chromosomes. Meanwhile, regarding the values of ρ in all instances, 32 chromosomes (45% of the 70 chromosomes) are at least 30% different on the structure, meaning the LS operator may also help to introduce extra diversity to the population.

4.8. Overall performance evaluation

This section evaluates the overall performance of pEA by comparing it with six state-of-the-art algorithms in the literature. The following explains the algorithms for comparison.

- GA1: BLS encoding-based GA (Kim *et al*, 2007b). Different from BLSGA used in Section 4.6, GA1 employs a greedy sweep operator after the evolution to further improve the quality of the best solution found by flipping each of the remaining 1's to 0 if it does not result into an infeasible solution.
- GA2: BTS encoding-based GA (Kim *et al*, 2007b). The same greedy sweep operator is applied at the end of evolution as in GA1.
- QEA1: Quantum-inspired evolutionary algorithm (QEA) (Xing *et al*, 2010). QEA maintains a population of quantum-bit chromosomes. Each chromosome is a probabilistic distribution model over the solution space. Each sampling on a chromosome results into a solution. Rotation angle step (RAS) and quantum mutation probability (QMP) are used to update each chromosome. QEA1 is based on the BLS encoding. For each chromosome, the RAS value is randomly generated and the QMP value is set according to the current fitness of the chromosome.
- QEA2: Another QEA proposed by Ji and Xing (2011). The main difference between QEA2 and QEA1 is that in QEA2 the RAS and QMP values of a chromosome are adjusted according to the current and previous fitness values of the chromosome.
- PBIL: Population-based incremental learning algorithm (Xing and Qu, 2011a). BLS encoding is used. PBIL maintains a real-valued probability vector (PV) which, when sampled, produces promising solutions with higher probabilities. At each generation, the statistic information of high-quality samples is used to update the PV. A restart scheme is introduced to help PBIL to escape from local optima.
- cGA: Compact genetic algorithm (Xing and Qu, 2012). Similar to PBIL, cGA also maintains a PV. However, the 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. Based on BLS encoding, cGA is featured by a restart scheme and a local search operator.
- pEA1: the path-oriented encoding EA. Note that LS operator is excluded. The performance of pEA1 will demonstrate the pure evolutionary search ability of the proposed algorithm.
- pEA2: pEA1 with LS operator, which indicates the overall performance of the proposed algorithm.

The population size is set to 20 for each algorithm. For GA1, we set $p_c = 0.8$ and $p_m = 0.006$. For GA2, we have $p_c = 0.8$ and $p_m = 0.012$. For QEA1, QEA2, PBIL and cGA, we adopt their best parameter settings (Xing *et al*, 2010; Ji and Xing, 2011; Xing and Qu, 2011a, 2012). For pEA\LS and pEA, we set $p_c = 0.9$ and $p_m = 1/d$, where d is the number of receivers.

Table 10 Comparisons of different algorithms

Networks	*Mean and SD*															
	*GA*1		*GA*2		*QEA*1		*QEA*2		*PBIL*		*cGA*		*pEA*1		*pEA*2	
	Mean	*SD*	*Mean*	*SD*	*Mean*	*SD*	*Mean*	*SD*	*Mean*	*SD*	*Mean*	*SD*	*Mean*	*SD*	*Mean*	*SD*
Fix-1	0.36	0.74	0.08	0.27	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	0.14	0.35	**0.00**	0.00
Fix-2	1.96	1.92	0.68	0.84	0.18	0.62	0.48	0.70	**0.00**	0.00	**0.00**	0.00	1.64	0.80	**0.00**	0.00
Fix-3	7.48	5.12	3.66	2.13	3.10	4.18	5.80	1.62	2.14	4.31	**0.00**	0.00	6.04	0.92	**0.00**	0.00
Fix-4	28.75	7.97	18.66	22.58	19.10	5.76	20.00	0.00	28.90	10.30	**0.00**	0.00	18.20	1.19	**0.00**	0.00
Rnd-1	0.52	0.88	0.44	0.50	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	0.04	0.19	**0.00**	0.00
Rnd-2	0.26	0.66	0.02	0.14	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rnd-3	0.44	0.83	0.02	0.14	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	0.04	0.19	**0.00**	0.00
Rnd-4	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rnd-5	2.78	2.71	1.16	0.61	0.46	0.50	0.48	0.54	0.04	0.28	0.04	0.19	0.02	0.14	**0.00**	0.00
Rnd-6	0.22	0.41	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
Rnd-7	1.58	0.92	1.36	0.66	0.66	0.47	0.58	0.53	0.38	0.60	0.22	0.41	0.34	0.47	**0.00**	0.00
Rnd-8	2.52	1.44	2.28	0.94	0.98	0.82	0.48	0.61	0.60	1.56	0.24	0.43	0.04	0.19	**0.00**	0.00
Rnd-9	2.82	1.22	2.34	1.34	1.64	0.98	1.94	1.16	0.06	0.23	0.04	0.19	0.86	0.35	**0.00**	0.00
Rnd-10	3.26	2.68	1.38	0.69	0.66	0.68	0.42	0.64	**0.00**	0.00	0.08	0.27	**0.00**	0.00	**0.00**	0.00

	SR (%)								*ACT (sec.)*							
	*GA*1	*GA*2	*QEA*1	*QEA*2	*PBIL*	*cGA*	*pEA*1	*pEA*2	*GA*1	*GA*2	*QEA*1	*QEA*2	*PBIL*	*cGA*	*pEA*1	*pEA*2
Fix-1	80	92	100	100	100	100	86	100	0.99	1.61	0.24	0.21	0.10	0.02	0.61	0.09
Fix-2	14	52	88	62	100	100	8	100	12.42	11.98	8.54	10.41	2.20	0.15	10.72	0.33
Fix-3	0	4	26	0	58	100	0	100	55.85	49.27	89.88	91.61	66.14	2.09	38.52	1.57
Fix-4	0	0	0	0	0	100	0	100	232.92	200.73	728.13	750.70	543.64	29.55	180.47	20.79
Rnd-1	62	56	100	100	100	100	96	100	2.95	3.30	0.73	0.50	0.29	0.23	0.39	0.16
Rnd-2	86	98	100	100	100	100	100	100	1.14	1.33	0.37	0.40	0.13	0.02	0.09	0.11
Rnd-3	76	98	100	100	100	100	96	100	5.13	5.07	0.68	0.75	0.23	0.06	0.77	0.27
Rnd-4	100	100	100	100	100	100	100	100	3.19	3.13	0.57	0.81	0.26	0.16	0.18	0.23
Rnd-5	4	10	54	54	98	96	98	100	16.57	14.52	13.82	14.38	6.09	3.14	2.85	0.63
Rnd-6	78	100	100	100	100	100	100	100	3.54	3.34	0.72	0.84	0.17	0.03	0.20	0.23
Rnd-7	8	8	34	44	68	78	66	100	24.13	20.78	24.35	22.52	24.29	6.83	11.31	2.10
Rnd-8	2	0	30	58	82	76	96	100	38.37	30.89	38.04	31.47	27.43	20.11	10.00	0.95
Rnd-9	4	8	14	10	94	96	14	100	62.46	50.73	73.73	73.94	47.29	16.40	38.86	1.93
Rnd-10	4	6	46	64	100	92	100	100	71.25	55.46	64.12	52.39	31.81	17.42	9.01	1.15

The comparison results are collected in Table 10, where the best results in mean are in bold. First, we analyse the data in Mean and SR for each algorithm. It can be seen that pEA2 always performs the best in each instance while cGA is the second best. The third best algorithm is PBIL. Compared with QEA1 and QEA2, PBIL performs better in six instances (see Fix-2,3 and Rnd-5,7,9,10) and worse in two instances (see Fix-4 and Rnd-8). The comparison of pEA1 and pEA2 illustrates that the LS operator helps to improve the overall performance of the proposed algorithm. In some cases the improvement is substantial, for example the mean and SR in instances Fix-2,3,4. When comparing pEA1 with the existing algorithms, we can see that in fix networks, pEA1 has similar performance with GA1. In random networks, pEA1 gains similar performance with PBIL except for instances Rnd-8,9 and illustrates better performance than GAs and QEAs in most instances.

Next, we compare the ACT of the algorithms. Before analysing the data, we divide the 14 instances into two groups according to their PIS values (see Section 4.2). Those with a PIS value less than 100% belong to the first group (called easy instances) while the rest belong to the second group (called hard instances). Easy instances includes Fix-1 and Rnd-1,2,3,4,6 while hard instances are Fix-2,3,4 and Rnd-5,7,8,9,10. Regarding easy instances, one can find that more than half of the state-of-the-art algorithms (GA1, GA2, QEA1, QEA2, PBIL, and cGA) can find an optimal solution with a successful ratio of 100%. As for hard instances, most of the state-of-the-art algorithms have a lower successful ratio than 100%. In easy instances, most of algorithms can obtain an optimal solution within a short time (eg less than 1 s). However, in each hard instance, the ACT spent by each algorithm differs significantly. In easy instances, QEA1, QEA2, PBIL, cGA, pEA1 and pEA2 all consume similar ACT (ie less than 1 s) while GA1 and GA2

Table 11 *t*-test results for comparing different algorithms

Networks	*Fix*-1	*Fix*-2	*Fix*-3	*Fix*-4	*Rnd*-1	*Rnd*-2	*Rnd*-3	*Rnd*-4	*Rnd*-5	*Rnd*-6	*Rnd*-7	*Rnd*-8	*Rnd*-9	*Rnd*-10
pEA2↔GA1	+	+	+	∼	+	+	+	+	+	+	+	+	+	+
pEA2↔GA2	+	+	+	+	+	∼	∼	∼	+	∼	+	+	+	+
pEA2↔QEA1	∼	+	+	+	∼	∼	∼	∼	+	∼	+	+	+	+
PEA2↔QEA2	∼	+	+	+	∼	∼	∼	∼	+	∼	+	+	+	+
pEA2↔PBIL	∼	∼	+	+	∼	∼	∼	∼	∼	∼	+	+	∼	∼
pEA2↔cGA	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼	+	+	∼	+
pEA2↔pEA1	+	+	+	+	∼	∼	∼	∼	∼	∼	+	∼	+	∼
pEA1↔GA1	∼	∼	∼	+	+	+	+	∼	+	+	+	+	+	+
pEA1↔GA2	∼	−	−	∼	+	∼	∼	∼	+	∼	+	+	+	+
pEA1↔QEA1	−	−	−	∼	∼	∼	∼	∼	+	∼	+	+	+	+
pEA1↔QEA2	−	−	∼	+	∼	∼	∼	∼	+	∼	+	+	+	+
pEA1↔PBIL	−	−	−	+	∼	∼	∼	∼	∼	∼	∼	+	−	∼
pEA1↔cGA	−	−	−	−	∼	∼	∼	∼	∼	∼	∼	+	−	+

Note: The result of comparison between Algorithm1↔Algorithm2 is shown as '+', '−', or '∼' when the former is significantly better than, significantly worse than, or statistically equivalent to the latter, respectively.

are the worst two. In hard instances, pEA2 and cGA are the two fastest algorithms. Besides, the former costs significantly less time than the latter in instances Fix-3,4 and Rnd-5,8,9,10. pEA1 is the third fastest algorithm. The difference between pEA1 and pEA2 indicates the effectiveness of the LS operator in reducing the computational time.

Regarding the overall performance in Table 10, we see that pEA2 is the best among the eight algorithms. Besides, pEA1 has similar performance with GA1 in fix networks and PBIL in random networks, respectively. Meanwhile, the LS operator has a positive impact on improving the overall performance of the proposed algorithm. To further support the finding, we show the *t*-test results comparing pEA2 and pEA1 with the others in Table 11.

5. Conclusions

This paper investigates the network coding resource minimization problem and develops a path-oriented encoding evolutionary algorithm (pEA) based on a new encoding approach. Different from the existing EAs which are based on the BLS or BTS encodings, the new EA is based on path-oriented encoding. Each chromosome consists of a number of BUs, each of which contains a set of link-disjoint paths from the source to the same receiver. In accordance to the new encoding approach, we develop the associated initialization, crossover and two mutation operators in the proposed EA. It is observed that between the two proposed mutation operators, the greedy mutation is more likely to result into a better performance than the ordinary mutation. Besides, a problem-specific local search operator is also developed to improve the solution quality. The simulation results show that the proposed pEA outperforms six existing state-of-the-art algorithms regarding the best solutions obtained and the computational time consumed, due to the new path-oriented encoding and the associated operators designed accordingly.

Acknowledgements—This work was supported in part by the China Scholarship Council, The University of Nottingham, National Natural Science Foundation of China (Grant No. 71001055) and Zhejiang Provincial Natural Science Foundation (Grant No. Y1100132).

References

Ahlswede R, Cai N, Li SYR and Yeung RW (2000). Network information flow. *IEEE Transactions on Information Theory* **46**(4): 1204–1216.

Ahn CW (2011). Fast and adaptive evolutionary algorithm for minimum-cost multicast with network coding. *Electronics Letters* **47**(12): 700–701.

Ahn CW and Ramakrishna RS (2002). A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Transactions on Evolutionary Computation* **6**(6): 566–579.

Cheng H and Yang S (2008). A genetic-inspired joint multicast routing and channel assignment algorithm in wireless mesh networks. In: *Proceedings of the 2008 UK Workshop on Computational Intelligence*, Leicester, UK: 2008 UK Workshop on Computational Intelligence. pp 159–164.

Cheng H and Yang S (2010a). Multi-population genetic algorithms with immigrants scheme for dynamic shortest path routing problems in mobile ad hoc networks. In: C Di Chio *et al*. (eds), EvoApplications 2010, Part I, LNCS 6024, Istanbul, Turkey. Springer-Verlag: Berlin, Heidelberg, pp 562–571.

Cheng H and Yang S (2010b). Genetic algorithms with immigrants schemes for dynamic multicast problems in mobile ad hoc networks. *Engineering Applications of Artificial Intelligence* **23**(5): 806–819.

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

Ji Y and Xing H (2011). A memory-storable quantum-inspired evolutionary algorithm for network coding resource minimization. *Evolutionary Algorithms*: 363–380.

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, MA.

Kim M *et al* (2007a). Evolutionary approaches to minimizing network coding resources. In: *Proceedings of 26th IEEE International Conference on Computer Communications (INFOCOM2007)*, Anchorage, AK, pp 1991–1999.

Kim M, Aggarwal V, Reilly VO, Médard M and Kim W (2007b). Genetic representations for evolutionary minimization of network coding resources. In: *Proceedings of EvoWorkshops 2007*, LNCS, Vol. 4448, Valencia, Spain, pp 21–31.

Langberg M, Sprintson A and Bruck J (2006). The encoding complexity of network coding. *IEEE Transactions on Information Theory* **52**(6): 2386–2397.

Li SYR, Yeung RW and Cai N (2003). Linear network coding. *IEEE Transactions on Information Theory* **49**(2): 371–381.

Luong HN, Nguyen HTT and Ahn CW (2012). Entropy-based efficiency enhancement techniques for evolutionary algorithms. *Information Sciences* **188**: 100–120.

Marczewski E and Steinhaus H (1958). On a certain distance of sets and the corresponding distance of functions. *Colloquium Mathematicum* **6**(1): 319–327.

Miller CK (1998). *Multicast Networking and Applications*. Pearson Education: Toledo, OH.

Mitchell M (1996). *An Introduction to Genetic Algorithms*. MIT Press: Cambridge, MA.

Oh S, Ahn CW and Ramakrishna RS (2006). A genetic-inspired multicast routing optimization algorithm with bandwidth and end-to-end delay constraints. In: I King *et al.* (eds), ICONIP 2006, Part III, LNCS 4234. Springer-Verlag: Berlin Heidelberg 2006, pp 807–816.

Palmer CC and Kershenbaum A (1994). Representing trees in genetic algorithms. In: *Proceedings of the First IEEE Conference on Evolutionary Computation*, Orlando, FL, pp 379–384.

Siregar JH, Zhang Y and Takagi H (2005). Optimal multicast routing using genetic algorithm for WDM optical networks. *IEICE Transactions on Communications* **E88-B**(1): 219–226

Walpole RE, Myers RH, Myers SL and Ye K (2007). *Probability and Statistics for Engineers and Scientists*. Pearson Education: Boston, MA.

Xing H and Qu R (2011a). A population based incremental learning for network coding resources minimization. *IEEE Communications Letters* **15**(7): 698–700.

Xing H and Qu R (2011b). A population based incremental learning for delay constrained network coding resource minimization. In: *Proceedings of Evo Applications 2011*, Turin, Italy, pp 51–60.

Xing H and Qu R (2012). A compact genetic algorithm for the network coding based resource minimization problem. *Applied Intelligence* **36**(4): 809–823.

Xing H and Qu R (2013). A nondominated sorting genetic algorithm for bi-objective network coding based multicast routing problems. *Information Sciences* **233**: 36–53.

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-International Journal of Electronics and Communications* **64**(12): 1105–1113.

Yang S and Yao X (2005). Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing* **9**(11): 815–834.

Yang S, Cheng H and Wang F (2010). Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile Ad Hoc networks. *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews* **40**(1): 52–63.

Received 28 June 2012;
accepted 31 May 2013 after two revisions