Appl Intell (2012) 36:809–823
DOI 10.1007/s10489-011-0298-8

A compact genetic algorithm for the network coding based resource minimization problem

Huanlai Xing · Rong Qu

Published online: 17 May 2011

Abstract In network coding based data transmission, intermediate nodes in the network are allowed to perform mathematical operations to recombine (code) data packets received from different incoming links. Such coding operations incur additional computational overhead and consume public resources such as buffering and computational resource within the network. Therefore, the amount of coding operations is expected to be minimized so that more public resources are left for other network applications.

In this paper, we investigate the newly emerged problem of minimizing the amount of coding operations required in network coding based multicast. To this end, we develop the first elitism-based compact genetic algorithm (cGA) to the problem concerned, with three extensions to improve the algorithm performance. First, we make use of an all-one vector to guide the probability vector (PV) in cGA towards feasible individuals. Second, we embed a PV restart scheme into the cGA where the PV is reset to a previously recorded value when no improvement can be obtained within a given number of consecutive generations. Third, we design a problem-specific local search operator that improves each feasible solution obtained by the cGA. Experimental results demonstrate that all the adopted improvement schemes contribute to an enhanced performance of our cGA. In addition, the proposed cGA is superior to some existing evolutionary algorithms in terms of both exploration and exploitation simultaneously in reduced computational time.

H. Xing (✉) · R. Qu
The Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, The University of Nottingham, Nottingham, NG8 1BB, UK
e-mail: hxx@cs.nott.ac.uk

R. Qu
e-mail: rxq@cs.nott.ac.uk

Keywords Compact genetic algorithm · Estimation of distribution algorithm · Multicast · Network coding

1 Introduction

Network coding represents a generalized routing scheme in communications, and has been attracting increasing research attention in both information theory and computer science since its introduction in 2000 [1]. As a newly emerged paradigm, it brings a lot of benefits to communication networks in terms of increased throughput, balanced network payload, energy saving, security, robustness against link failures, and so on [2–7]. Instead of simply replicating and forwarding data packets at the network layer, network coding allows any intermediate node (i.e. router), if necessary, to perform arbitrary mathematical operations to recombine (i.e. code) data packets received from different incoming links. By doing so, the maximized multicast throughput bounded by the MAX-FLOW MIN-CUT theorem can always be obtained [2].

Figure 1 shows an example of the superiority of network coding over traditional routing in terms of the maximum multicast throughput achieved [8]. In the network of 7 nodes and 9 links in Fig. 1(a), s is the single source, and y and z are two sinks. Each direct link has a capacity of one bit per time unit. According to the MAX-FLOW MIN-CUT theorem, we know that the minimum cut $C_{\min}$ between s and y (or between s and z) is two bits per time unit, so is the maximum multicast throughput from s to y and z. However, only 1.5 bits per time unit can be achieved as the multicast throughput if traditional routing is used. This is because link $w \rightarrow x$ could only forward one bit (a or b) at a time to node x, and thus y and z cannot simultaneously receive two bits, as indicated in Fig. 1(b). In Fig. 1(c) where network coding

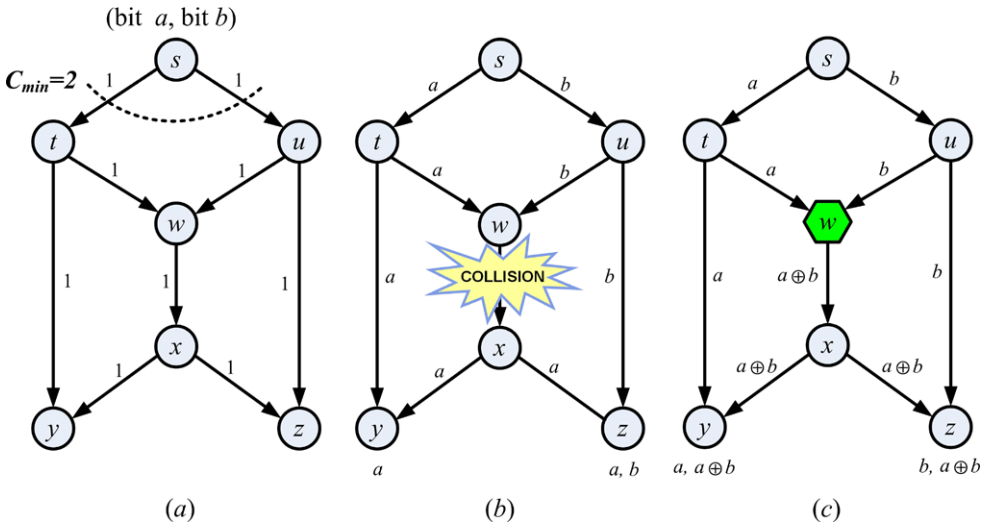

Fig. 1 Traditional routing vs. network coding [8]. (**a**) The example network. (**b**) Traditional routing scheme. (**c**) Network coding scheme

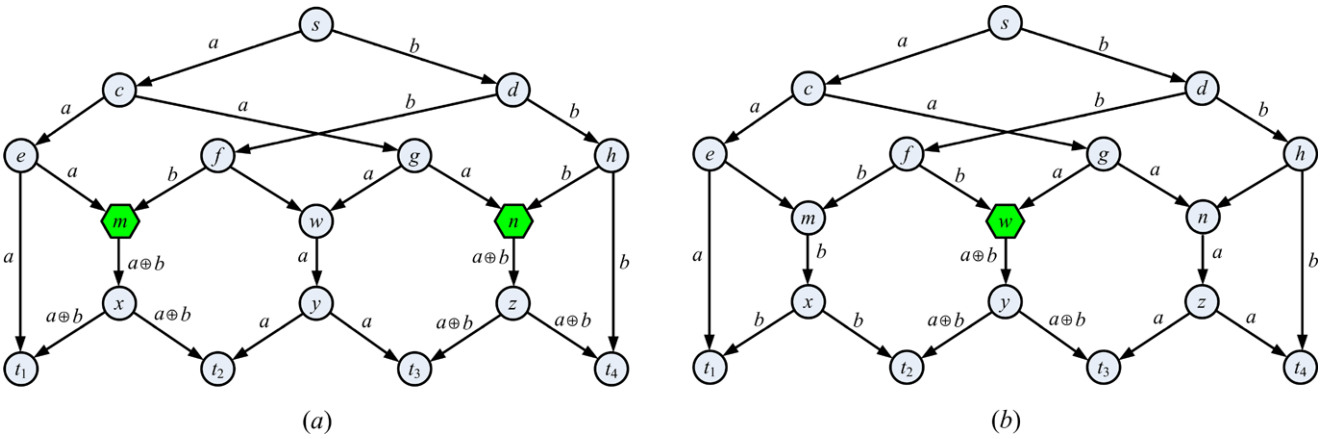

Fig. 2 Two different network-coding-based data transmission schemes. (**a**) Scheme A with two coding nodes. (**b**) Scheme B with only one coding node

is applied, node w is allowed to recombine the two bits it receives from t and u into one bit $a \oplus b$ (symbol $\oplus$ here represents the Exclusive-OR operation) and to output $a \oplus b$ to node x. In this way, y and z are able to receive $\{a, a \oplus b\}$ and $\{b, a \oplus b\}$ respectively, and thus two bits information is available at each sink. Meanwhile, by calculating $a \oplus (a \oplus b)$ and $b \oplus (a \oplus b)$, y and z can then recover b and a, respectively.

In most of the previous research in network coding, coding is performed at all coding-possible nodes. However, to obtain an expected multicast throughput, coding may only be necessary at a subset of those nodes [9–11]. Figure 2 illustrates two network-coding-based data transmission schemes that could both achieve the maximum multicast throughput. Source s expects to transmit two bits (a and b) to four sinks, t_1, t_2, t_3 and t_4. Scheme A adopts two coding-possible nodes, namely node m and node n, as shown in Fig. 2(a). Nevertheless, the same throughput can also be obtained by scheme B in Fig. 2(b), where coding only occurs at node w. Due to the mathematical operations involved, network coding not only incurs additional cost such as computational overhead and transmission delay, but also consumes public resources, e.g. buffering and computational resources [12]. It is therefore important that the number of coding operations is kept minimized while the benefits of network coding are warranted. Unfortunately, this problem is NP-hard [9–11].

Although a large amount of research has been conducted on multicast routing problems by using advanced algorithms including evolutionary algorithms and local search based algorithms [13–17], a limited number of algorithms have been proposed in the literature of network coding based multicast. Most of these algorithms are based on either greedy methods or evolutionary algorithms. Langberg et al. [12] and Fragouli et al. [18] proposed different network decomposition methods and two greedy algorithms to minimize coding operations. However, the optimization of these algorithms depends on the traversing order of links. An inappropri-

ate link traversal order leads to a deteriorated performance. Kim et al. investigated evolutionary approaches to minimize the required network coding resources [9–11]. In [9], a genetic algorithm (GA) working in an algebraic framework has been put forward. However, it is applied to acyclic networks only. This has been extended to a distributed GA to significantly reduce the computational time in [10]. In [11], the authors compare and analyse GAs with two different genotype encoding approaches, i.e. the binary link state (BLS) and the binary transmission state (BTS). Simulations show that compared to BLS encoding, BTS encoding has much smaller solution space and leads to better solutions. Besides, their GA-based algorithms perform outstandingly better than the two greedy algorithms in [12] and [18] in terms of the best solutions achieved. Nevertheless, as we observed in our present work, GAs (e.g. [11]) have still shown to be weak in global exploration, even though a greedy sweep operator follows the evolution to further improve the best individual. In our previous work [8], an improved quantum-inspired evolutionary algorithm (QEA) has been developed to minimize the amount of coding operations. Simulation results demonstrate that the QEA outperforms simple GAs in many aspects including fast convergence. However, we observe in this paper that the improved QEA sometimes finds decent solutions at the cost of additional computational time. Recently, we also put forward a population based incremental learning (PBIL) to find the optimal amount of coding operations [19]. However, its main concern is how to apply network coding in delay sensitive applications. An extended compact genetic algorithm has thus been developed in this work to solve the highly constrained problems being concerned.

As one of estimation of distribution algorithms (EDA) [20–22], the compact genetic algorithm (cGA) was first introduced in 1999 by Harik et al. [23]. Whereas the simple genetic algorithm (sGA) maintains a population of solutions, cGA simply employs a probability vector (PV) while still retaining the order-one behavior (i.e. problem can be solved to optimality by combining only order-one schemata [23]) of the sGA with a uniform crossover. Contrary to sGA, cGA is much faster and efficient, and requires far less memory so that significant amounts of computational time and memory are saved. Hence, cGA has drawn an increasing research attention and been successfully applied to a number of optimization problems including evolvable hardware implementation [24, 25], multi-FPGA partitioning [26], image recognition [27], TSK-type fuzzy model [28] and so on. Unfortunately, cGA is not always powerful, especially to complex optimization problems, due to the assumption that variables in any given problem are independent [29].

In this paper, we investigate the first elitism-based cGA to the minimization problem of coding operations in network coding based multicast. In our cGA, three novel schemes have been developed to improve the optimization performance of cGA. The first scheme is to, by using an all-one vector, adjust the PV in such a way that feasible individuals appear with higher probabilities. This scheme not only warrantees the cGA with a feasible elite individual at the beginning of evolution but also allows the PV to generate feasible individuals with increasingly higher probabilities. The second scheme is a PV restart scheme to reset the PV when the solution found cannot be improved within a given number of consecutive generations. This scheme stops ineffective evolution and helps to increase the chance to hit an optimal solution. In the third scheme, a local search operator is devised to exploit the neighborhood of each feasible solution so that the local exploitation of our cGA is, to a large extent, enhanced. Simulation experiments have been conducted over a number of fixed and randomly generated multicast scenarios. Results demonstrate that all the adopted schemes are effective and the proposed cGA outperforms existing evolutionary algorithms in obtaining optimal solutions within reduced computational time.

2 Problem description

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 [2]. 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 the 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 (a capacity of R units) is achievable if there is a transmission scheme that enables each sink t_k, $k = 1, \ldots, d$, to receive the information at the data rate R [9–11]. We assume each link has a unit capacity, and a path from s to t_k thus has a unit capacity. If we manage to set up R link-disjoint paths $\{P_1(s, t_k), \ldots, P_R(s, t_k)\}$ from s to each sink $t_k \in T$, we make the data rate R achievable. In this work we consider the linear network coding scheme which is sufficient for multicast applications [2].

In this paper, a subgraph in G is called a *network coding based multicast subgraph* (NCM subgraph, denoted by $G_{NCM}(s, T)$) if there are R link-disjoint paths $P_i(s, t_k)$, $i = 1, \ldots, R$, from s to each sink t_k, $k = 1, \ldots, d$, in this subgraph. An intermediate node n_c is called a *coding node* if it performs a coding operation. Each coding node has at least one outgoing link, called *coding link*, if this link outputs the coded information. Take data transmission scheme in Fig. 1(c) as an example, its NCM subgraph and the paths that make up of this subgraph are shown in Fig. 3. The NCM subgraph is composed of four paths, i.e. $P_1(s, y)$, $P_2(s, y)$, $P_1(s, z)$ and $P_2(s, z)$, where paths to the same sink are link-disjoint. As we know, no coding is necessary at any intermediate node with only one incoming link. We refer to each

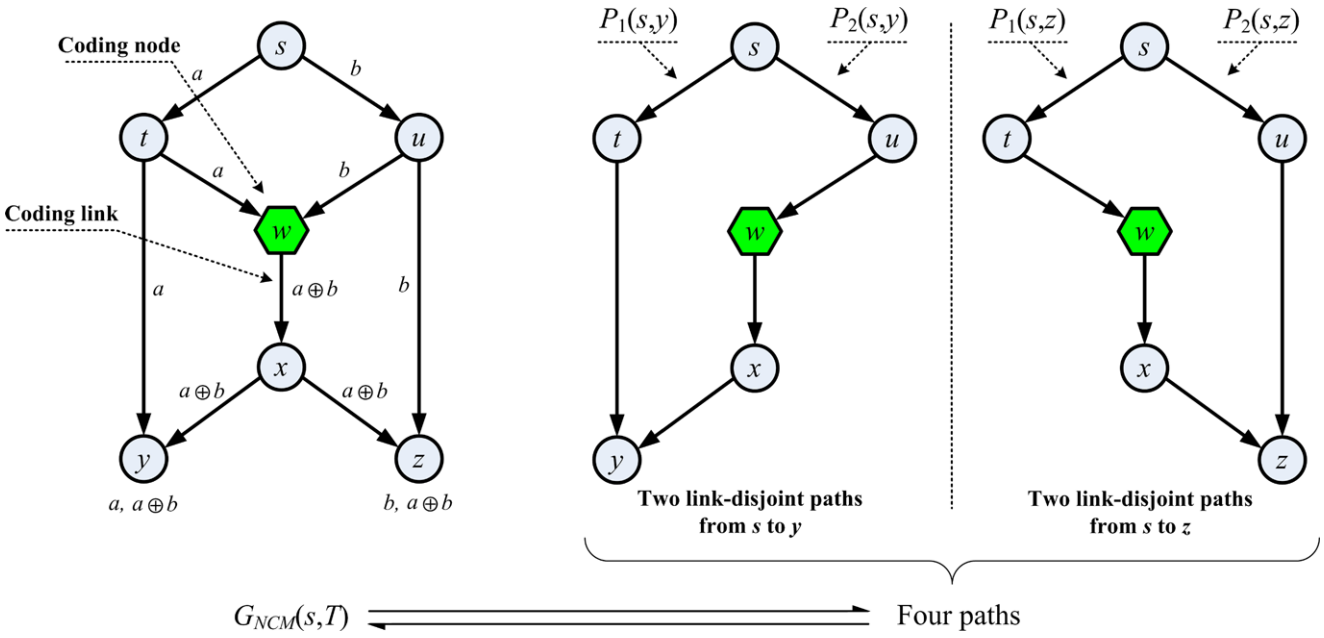

Fig. 3 An example of the NCM subgraph and the paths that make up of it

non-sink node with multiple incoming links as a *merging node* which can perform coding [10, 11]. We also refer to each outgoing link of a merging node as a *potential coding link*. To determine if a potential coding link of a merging node becomes a coding link, we just need to check if the information via this link is dependent on a number of incoming links of the merging node.

For a given multicast scenario (G, s, T, R), the number of coding links, rather than coding nodes, is more precise to indicate the total amount of coding operations [12]. We therefore investigate how to construct a NCM subgraph $G_{NCM}(s, T)$ with the minimal number of coding links while achieving the expected data rate. We define the following notations:

v_{ij}: a variable associated with the jth outgoing link of the ith merging node, $i = 1, \ldots, M$, $j = 1, \ldots, Z_i$, where M is the total number of merging nodes and the ith merging node has Z_i outgoing links. $v_{ij} = 1$ if the jth outgoing link of the ith node serves as a coding link; $v_{ij} = 0$ otherwise.

$n_{cl}(G_{NCM}(s, T))$: the number of coding links in a constructed NCM subgraph $G_{NCM}(s, T)$.

$R(s, t_k)$: the achievable rate from s to t_k.

R: the defined data rate (an integer) at which s expects to transmit information.

$P_i(s, t_k)$: the ith established path from s to t_k, $i = 1, \ldots, R$ in $G_{NCM}(s, T)$.

$W_i(s, t_k)$: the set of links of $P_i(s, t_k)$, i.e. $W_i(s, t_k) = \{e | e \in P_i(s, t_k)\}$.

Based on the above notations, we define in this paper the problem of network coding based resource minimization as to minimize the number of coding links while achieving a desired multicast throughput, shown as follows:

$$\text{Minimize:} \quad n_{cl}(G_{NCM}(s, T)) = \sum_{i=1}^{M} \sum_{j=1}^{Z_i} v_{ij} \tag{1}$$

$$\text{Subject to:} \quad R(s, t_k) \geq R, \quad \forall t_k \in T \tag{2}$$

$$\bigcap_{i=1}^{R} W_i(s, t_k) = \emptyset, \quad \forall t_k \in T \tag{3}$$

Objective (1) defines our problem as to minimize the number of coding links in the constructed NCM subgraph; Constraint (2) defines that the achievable data rate from s to each sink must be at least R so that we can set up R paths for each sink; Constraint (3) indicates that for an arbitrary t_k the R constructed paths $P_i(s, t_k)$, $i = 1, \ldots, R$, must have no common link so that each sink can receive information at rate R.

3 An overview of compact genetic algorithm

cGA is a variant of EDA, where its population is implicitly represented by a real-valued probability vector (PV). At

Standard cGA

1) **Initialization**
2) Set $t := 0$;
3) **for** $i = 1$ **to** L **do** set $P_i^t := 0.5$
4) **repeat**
5) Set $t := t + 1$;
6) // Generate two individuals from the PV
$X_a :=$ **generate** $(\boldsymbol{P}(t))$; $X_b :=$ **generate** $(\boldsymbol{P}(t))$;
7) // Let X_a and X_b compete
winner, loser := **compete** (X_a, X_b);
8) // The PV learns towards the winner
for $i = 1$ **to** L **do**
if winner(i) <> loser(i) **then**
if winner$(i) == 1$ **then** $P_i^t := P_i^t + 1/N$;
else $P_i^t := P_i^t - 1/N$;
9) **until** the PV has converged
10) **Output** the converged PV as the final solution

Fig. 4 Procedure of the standard cGA

each generation, only two individuals are sampled from the PV and a single tournament is performed between them, i.e. a winner and a loser are identified [23]. The PV is then adjusted and shifted towards the winner. As the cGA evolves, the PV converges to an explicit solution.

We denote the aforementioned PV at generation t by $\boldsymbol{P}(t) = \{P_1^t, \ldots, P_L^t\}$, where L is the length of each individual (see more details in Sect. 4). The value at each locus of $\boldsymbol{P}(t)$, i.e. $P_i^t, i = 1, \ldots, L$, is initialized as 0.5 so that initially all solutions in the search space appear with the same probability. Let winner(i) and loser(i), $i = 1, \ldots, L$, be the ith bit of the winner and the loser, respectively, and $1/N$ be the increment of the probability of the winning alleles after each competition between the winner and the loser, where N is an integer. Note that although cGA produces two individuals at each generation, it can mimic the convergence behavior of a sGA with a population size N [23]. The procedure of the standard cGA is presented in Fig. 4.

In this paper, the proposed cGA is based on the persistent elitist cGA (pe-cGA) introduced in [29]. Compared with the standard cGA, the procedure of pe-cGA is almost the same except the two steps, i.e. steps 6 and 7, in Fig. 4. Figure 5 illustrates steps 6 and 7 in pe-cGA [29], where two individuals are created at generation $t = 1$. In the following generations, only one new individual is created to compete with the winner from previous generations. The winner (the elite individual), on the other hand, is never changed as long as no better individual has been sampled from the PV.

6) // Generate one individual from the PV
if $t == 1$ **then**
$X_e :=$ **generate** $(\boldsymbol{P}(t))$; // initialize the elite individual
$X_{new} :=$ **generate** $(\boldsymbol{P}(t))$; // generate a new individual
7) // X_e and X_{new} compete and the winner inherits
winner, loser := **compete** (X_e, X_{new});
$X_e :=$ winner; // update the elite individual

Fig. 5 The different steps in pe-cGA [19]] compared with the standard cGA in Fig. 1

4 The proposed compact genetic algorithm

In this section, we first describe the individual representation and the fitness evaluation in our proposed cGA, based on which all the aforementioned improvement schemes are devised.

4.1 Individual representation and fitness evaluation

Encoding represents one of the most important key issues in designing efficient and effective evolutionary algorithms in many complex optimization problems, including the newly emerged coding resource minimization problem concerned in our work. To cater for the complex network structure in the problem studied here, we adopt the Graph Decomposition Method in [10, 11] to represent solutions and calculate the fitness of each individual in the cGA.

To detect the number of coding operations at each merging node in a given network topology G, a secondary graph G_D is created by decomposing each merging node in G into a number of nodes connected with additional links introduced. For the ith merging node with $In(i)$ incoming links and $Out(i)$ outgoing links, $In(i)$ nodes, $u_1, \ldots, u_{In(i)}$, referred to as incoming auxiliary nodes, and $Out(i)$ nodes, $w_1, \ldots, w_{Out(i)}$, referred to as outgoing auxiliary nodes, are created. The original ith merging node can thus be seen as decomposed into two sets of nodes. The jth incoming link of the ith original merging node is redirected to node u_j; and the kth outgoing link of the ith merging node is redirected to node w_k. Besides, a directed link $e(u_j, w_k)$ is inserted between each pair of nodes (u_j, w_k), $j = 1, \ldots, In(i)$, $k = 1, \ldots, Out(i)$.

In our cGA, we associate each binary bit of an individual X with one of the newly introduced links between those auxiliary nodes, e.g. $e(u_j, w_k)$ in G_D. A value '1' at a bit in X means its corresponding link exists in G_D, '0' otherwise. Each individual therefore corresponds to an explicit secondary graph G_D which may or may not provide a valid network coding based routing solution.

To evaluate a given individual X, we first check if X is feasible. For each sink $t_k, k = 1, \ldots, d$, we use the Goldberg

algorithm [30], a classical max-flow algorithm, to compute the max-flow between the source s and t_k in the corresponding G_D of X. As mentioned in Sect. 2, 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 found by the Goldberg algorithm between s and t_k. If all d max-flows are at least R, rate R is achievable and the individual X is feasible. Otherwise, the individual X is infeasible.

For each infeasible individual X, we set a sufficiently large fitness value Ψ to its fitness $f(X)$ (in this paper, $\Psi = 50$). If X is feasible, we first determine its corresponding NCM subgraph $G_{NCM}(s, T)$ and then calculate its fitness value. For each sink $t_k \in T$, we select R paths from the obtained link-disjoint paths from s to t_k (if the max-flow is R then we select all the link-disjoint paths), and therefore obtain in total $R \cdot d$ paths, e.g. $P_i(s, t_j)$, $i = 1, \ldots, R$, $j = 1, \ldots, d$. We map all the selected paths to G_D and obtain a $G_{NCM}(s, T)$ in which coding operations occur at the outgoing auxiliary nodes with two or more incoming links. The fitness value $f(X)$ can then be set to the number of coding links in the $G_{NCM}(s, T)$.

Note that our fitness evaluation is slightly different from the one in [10, 11] where authors only concern if the subgraph obtained can meet the data rate requirement, i.e. each constructed subgraph in [10, 11] potentially offers a data rate larger than required because the subgraph may have more link-disjoint paths than expected from the source to the sink. Our NCM subgraph provides the exact expected data rate, and thus is more likely to occupy less link and coding resources.

4.2 The use of an all-one vector

The problem concerned in our work is highly constrained within complex network structures, and thus infeasible solutions form a large proportion of the solution space. The PV hence may not be able to efficiently evolve with a limited number of feasible individuals, i.e. the optimization of cGA could be seriously weakened due to the lack of feasible individuals. Kim et al. [10, 11] noticed this problem and inserted an all-one vector, i.e. '11...1', into the initial population to warrantee that their GAs start with at least one feasible individual (the all-one vector ensures that all newly introduced links exist in G_D and guarantees a feasible NCM subgraph to be found). Recently, we also use all-one vectors to guide population based incremental learning (PBIL) towards feasible solutions [19]. The above methods both show to improve the optimization performance.

Inspired by the idea in [10, 11, 19], we simply set an all-one vector as the elite individual in the initialization to ensure that our cGA begins with at least one feasible individual. It is not hard to understand that individuals containing more 1s are more likely to be feasible for the problem concerned. The PV which gradually shifts towards the all-one vector thus gets increasingly higher chance to produce feasible individuals. This method shows to significantly accelerate the convergence speed of the cGA in the early stage of evolution (see Sect. 5 for details).

4.3 The probability vector restart scheme

In the literature of EDA, restart schemes have been introduced in population-based incremental learning (PBIL) approaches to avoid premature evolution and thus to enhance the global search capability of PBIL. The essence of these schemes is to restart (re-initialize) the search under certain restart conditions. For example, in applying PBILs to dynamic optimization problems, the PV can be reset as soon as the environment changes [31, 32]. Similarly, Ahn et al. [29] present an efficient nonpersistent elitist cGA (ne-cGA) where an elite individual is regenerated using the initialized PV, i.e. $P_i^t = 0.5$, $i = 1, \ldots, L$, when the current elite individual dominates in a predefined number of generations.

For the complex and difficult problem concerned in this paper, we design a PV restart scheme which replaces the current PV with a previously recorded PV when no better individual can be found within a predefined number of consecutive generations, i.e. g_c generations, where g_c is an integer. The principle to choose an appropriate recorded PV is that it should be able to generate feasible individuals with a high probability so that the cGA retains an effective evolution. On the other hand, the PV should have an appropriate probability distribution to maintain a good diversity, where the current elite individual has less chance to appear again in the evolution.

Let X_e denote the elite individual. The steps of the PV restart scheme in our paper are shown as follows:

(1) Record the PV that generates the first feasible individual during the evolution of cGA. For example, if the PV produced the first feasible individual at generation t_i, we record $\boldsymbol{P}(t_i)$ for the proposed restart scheme. Here, we assume during the evolution of cGA, the PV can produce feasible individuals.
(2) After $\boldsymbol{P}(t_i)$ is recorded, the restart scheme is launched. We set $counter = 0$ at generation t_i. Note that the initial value of $counter$ is -1, which implies the restart scheme has not started.
(3) If X_e is not changed in a new generation, set $counter = counter + 1$. If $counter = g_c$, which means X_e stays in cGA for g_c consecutive generations, set PV $= \boldsymbol{P}(t_i)$ and $counter = 0$. Note that the current X_e remains unchanged, and is likely to be changed once the PV is reset. On the other hand, once X_e is changed, set $counter = 0$.

This scheme shows to effectively improve the global exploration capability of our cGA (see Sect. 5).

4.4 The local search operator

As mentioned in Sect. 4.1, each feasible individual corresponds to a secondary graph G_D based on which a NCM subgraph $G_{NCM}(s,T)$ can be found in the fitness evaluation by using the Goldberg algorithm in [30]. However, one feature of G_D is that the NCM subgraph found may not be unique, i.e. we could possibly find alternative feasible NCM subgraphs from the given G_D. We call these feasible NCM subgraphs the neighbors of $G_{NCM}(s,T)$ in G_D. The better the NCM subgraph found (i.e. the NCM subgraph requires less coding operations), the higher the quality of the corresponding solution to the problem, and the faster the convergence of the algorithm.

In order to find a better NCM subgraph in G_D, we propose a local search operator (L-operator), which starts from $G_{NCM}(s,T)$, to explore the neighbors of the $G_{NCM}(s,T)$ and find hopefully a neighbor with less coding operations.

Assume $G_{NCM}(s,T)$ has h coding nodes, i.e. $n_1,\ldots,n_h$, where the ith coding node n_i has $In(i)$ incoming links. We denote by e_{ij} the jth incoming link of n_i in $G_{NCM}(s,T)$. Obviously, there is also an identical e_{ij} in G_D since $G_{NCM}(s,T) \subseteq G_D$. The procedure of the L-operator is shown as follows:

(1) Set $G_D^{temp} = G_D$; $G_{NCM}^{temp} = G_{NCM}(s,T)$; $i=1$; $j=1$;
(2) Remove the link e_{ij} from G_D^{temp}. Use the Goldberg algorithm [30] to calculate the max-flow between s and each sink in G_D^{temp}. If the d max-flows are at least R, go to step 3. Otherwise, reinsert the link e_{ij} to G_D^{temp} and go to step 4.
(3) For each sink t_k, randomly select R paths from the obtained link-disjoint paths from s to t_k (if there are R paths we then select all of them), and map all the selected paths to G_D^{temp} to obtain a new NCM subgraph $G_{NCM}^{new}(s,T)$. If the number of coding links in $G_{NCM}^{new}(s,T)$ is less than that in G_{NCM}^{temp}, set $G_{NCM}^{temp} = G_{NCM}^{new}(s,T)$; otherwise, reinsert the link e_{ij}.
(4) If $j = In(i)$, proceed to the next coding node, i.e. set $i = i+1$ and go to step 5; otherwise, proceed to the next incoming link of the same coding node, i.e. set $j = j+1$, and go to step 2.

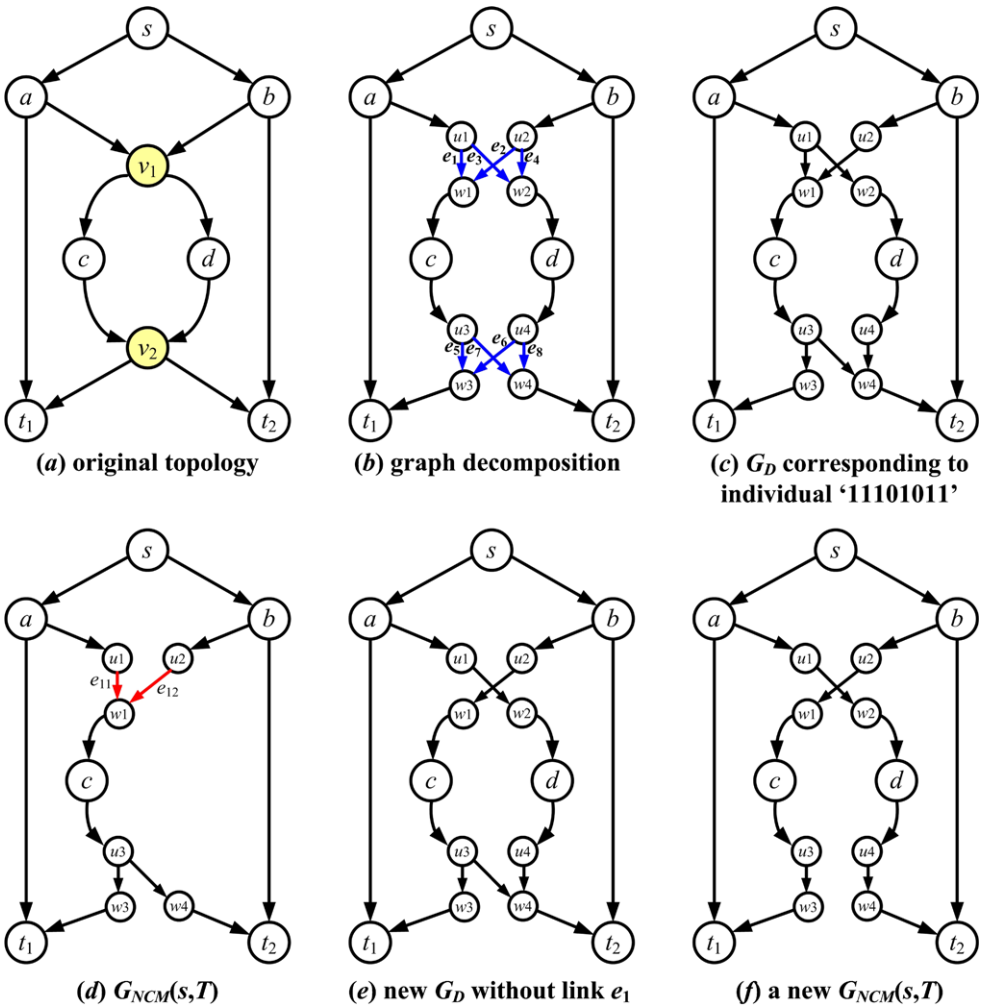

Fig. 6 An example of the graph decomposition and local search procedure

(5) If $i = h+1$, stop the procedure and output G_{NCM}^{temp}; otherwise, proceed to the first incoming link of the ith coding node, i.e. set $j = 1$, and go to step 2.

Figure 6 shows an example of the graph decomposition and the local search procedure. The original graph with the source s and sinks t_1 and t_2, as shown in Fig. 6(a), has two merging nodes, i.e. v_1 and v_2. In Fig. 6(b), v_1 and v_2 are decomposed into two groups of auxiliary nodes with newly introduced links, i.e. $e_1, \ldots, e_8$. We assume the ith bit of an individual is associated with link $e_i, i = 1, \ldots, 8$ (see Sect. 4.1). Given an individual '11101011', its corresponding secondary graph G_D is shown in Fig. 6(c). Based on G_D, we can obtain a NCM subgraph $G_{NCM}(s, T)$ with only one coding node $w1$, as shown in Fig. 6(d), where, obviously, links e_{11} and e_{12} are two incoming links of $w1$. In Fig. 6(e), the L-operator removes e_{11} from G_D and a new G_D is obtained. Based on the new G_D, we can obtain a better NCM subgraph shown in Fig. 6(f) as this subgraph is coding-free. It can be seen that removing e_{12} from Fig. 6(e) produces an infeasible NCM subgraph thus e_{12} is retained. Since all incoming links of coding nodes are checked, the L-operator stops and outputs the NCM subgraph shown in Fig. 6(f). In this example, the fitness of '11101011' is thus set to zero.

In our cGA, the L-operator is applied to improve the NCM subgraph $G_{NCM}(s, T)$ of each feasible individual X. The number of coding links in the resulting G_{NCM}^{temp} is returned as the fitness of the improved individual.

The L-operator reveals to be quite effective to improve the quality of solutions obtained by our cGA (see Sect. 5).

4.5 The overall procedure of the proposed cGA

The pseudo-code of the proposed cGA is shown in Fig. 7 with the following three extensions for solving the problem concerned: (1) In the initialization the elite individual is set to an all-one vector; (2) the PV is reset when the elite individual is not changed within a number of consecutive generations; (3) a local search operator is integrated to exploit the neighbours of each feasible NCM subgraph found.

The genotype encoding approach used in the proposed cGA 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 [10, 11] to the outgoing link.

Different from pe-cGA that creates and evaluates the elite individual at generation $t = 1$ as shown in Fig. 5, we create and evaluate the elite individual in the initialization (where $t = 0$). The elite individual X_e is set to an all-one vector to ensure our cGA begins with a feasible individual. In each generation, an individual X is sampled from the PV, and its feasibility is checked by the fitness evaluation. If X corresponds to a G_D where a feasible NCM subgraph can be found, we use the L-operator to search the neighbors of the NCM subgraph and obtain hopefully a new and better NCM subgraph. The number of coding links in the new NCM subgraph is set to the fitness of X, i.e. $f(X)$. Otherwise, a sufficiently large fitness value $\Psi(= 50)$ is set to $f(X)$ for an infeasible solution; If X is the first feasible individual, we record the PV from which X is created, and launch the PV restart scheme by setting $counter = 0$ in step 10. The restart scheme is triggered when no better individual appears within g_c generations, as shown in step 11. In step 12, we update

1) **Initialization**
2) Set $t := 0$; $counter := -1$; // see section 4.3
3) **for** $i = 1$ **to** L **do** set $P_i^t := 0.5$ // initialize PV
4) // Initialize the elite individual with an all-one vector
   $X_e := 11\ldots1$; // see section 4.2
5) // Evaluate the elite individual
   $f(X_e) :=$ **evaluate** (X_e); // see section 4.4
6) **repeat**
7) Set $t := t + 1$;
8) // Generate one individual from the PV
   $X :=$ **generate** $(\boldsymbol{P}(t))$; // X is sampled from $\boldsymbol{P}(t)$
9) // Evaluate the individual
   $f(X) :=$ **evaluate** (X); // see section 4.4
10) // Record the PV for the restart scheme
    if *X is the first feasible individual* **then**
      $counter := 0$; $PV_{record} := \boldsymbol{P}(t)$; // see section 4.3
11) // The PV restart scheme
    if $f(X_e) \le f(X)$ && $counter \ge 0$ **then**
      $counter := counter + 1$;
      if $counter == g_c$ **then**
        $\boldsymbol{P}(t) := PV_{record}$; $counter = 0$; // see section 4.3
12) // Record better individuals
    if $f(X_e) > f(X)$ **then**
      $X_e := X$; $f(X_e) := f(X)$;
      if $counter > 0$ **then**
        $counter := 0$;
13) // The PV learns towards the elite individual
    for $i = 1$ **to** L **do**
      if $X_e(i) <> X(i)$ **then**
        if $X_e(i) == 1$ **then** $P_i^t := P_i^t + 1/N$;
        else $P_i^t := P_i^t - 1/N$;
14) **until** *the termination condition is met*

Fig. 7 Procedure of the proposed cGA

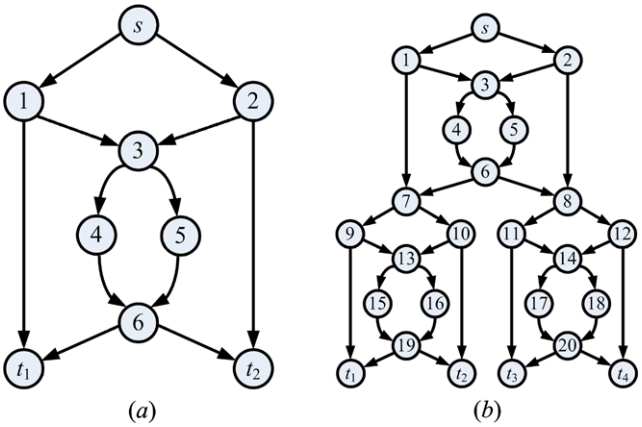

Fig. 8 An example of the n-copies network. (**a**) The original network (**b**) the 3-copies network

the elite individual X_e if the new X is better than X_e. Then, the PV is shifted towards X_e in step 13.

The procedure terminates subject to two conditions: (1) a feasible NCM subgraph without coding is obtained, or (2) the algorithm reaches a pre-defined number of generations.

5 Performance evaluation

We first investigate the effects of the three extensions in cGA, i.e. the use of an all-one vector, the PV restart scheme and the local search operator. Experiments have been firstly conducted on two directed networks, i.e. 3-copies and 20-nodes, before an overall evaluation of the algorithm on eleven more multicast scenarios. Similar to the structures of n-copies in [11], our n-copies networks are generated based on the network shown in Fig. 8(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. Figure 8 shows the original network and the 3-copies network with source s and sinks t_1, t_2, t_3 and t_4. On the other hand, the 20-node network (with 20 nodes, 37 links and 5 sinks) is randomly created, where the data rate is set to 3. The increment of each winning allele is set to 0.05 in the cGAs which mimics the performance of sGA with a population size of 20 ($= 1/0.05$). All experimental results are collected by running each algorithm 50 times.

5.1 The effect of the all-one vector

As mentioned above, the all-one vector not only enables the cGA to begin with a feasible individual but also adjusts the PV to produce feasible individuals with increasingly higher probabilities. In order to evaluate the performance of the all-one vector, we compare the following three variants of the cGA on the 3-copies and 20-node networks:

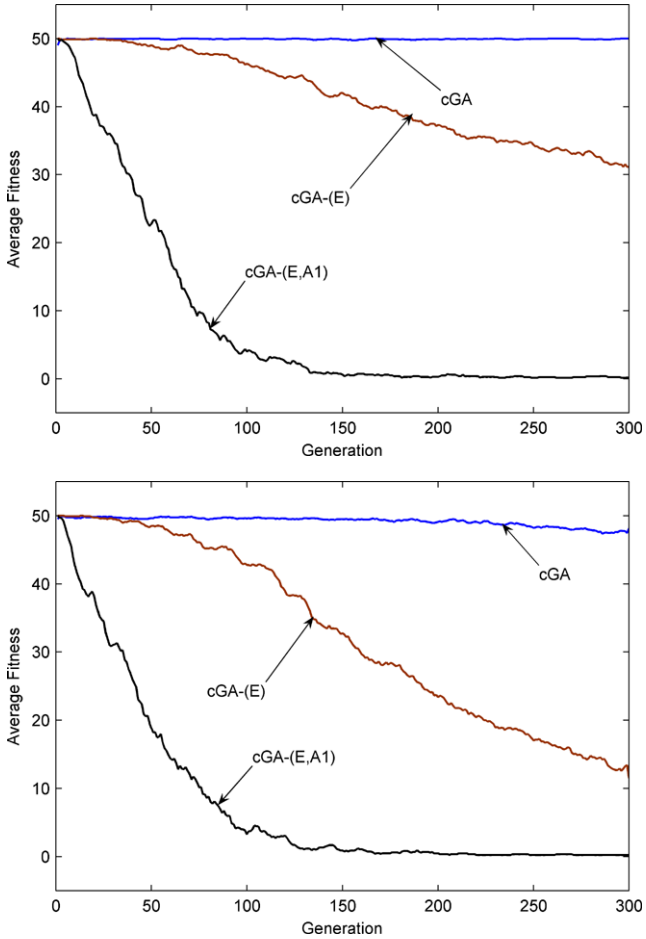

Fig. 9 Average fitness vs. generations in variants of cGA. (**a**) the 3-copies network (**b**) the 20-node network

- cGA: the cGA introduced in [23] (see Sect. 3).
- cGA-(E): cGA with persistent elitism scheme in [29].
- cGA-(E,A1): cGA-(E) with the use of the all-one vector (see Sect. 4.2).

We compare the above algorithms on the following four evaluation criteria:

- The evolution of the average fitness. The termination condition here is a pre-defined number of generations, i.e. algorithms stop after 300 generations.
- The successful ratio of finding a coding-free (i.e. with no coding) NCM subgraph in 50 runs. Note that the definition of the successful ratio in our experiments is different from that in [11]. In [11], the authors are concerned with the successful ratio of finding a feasible NCM subgraph in a number of runs.
- The average best fitness over 50 runs and the standard deviation.
- The average termination generation over 50 runs. The termination condition here is a coding-free NCM subgraph has been found or the algorithms have reached 300 generations.

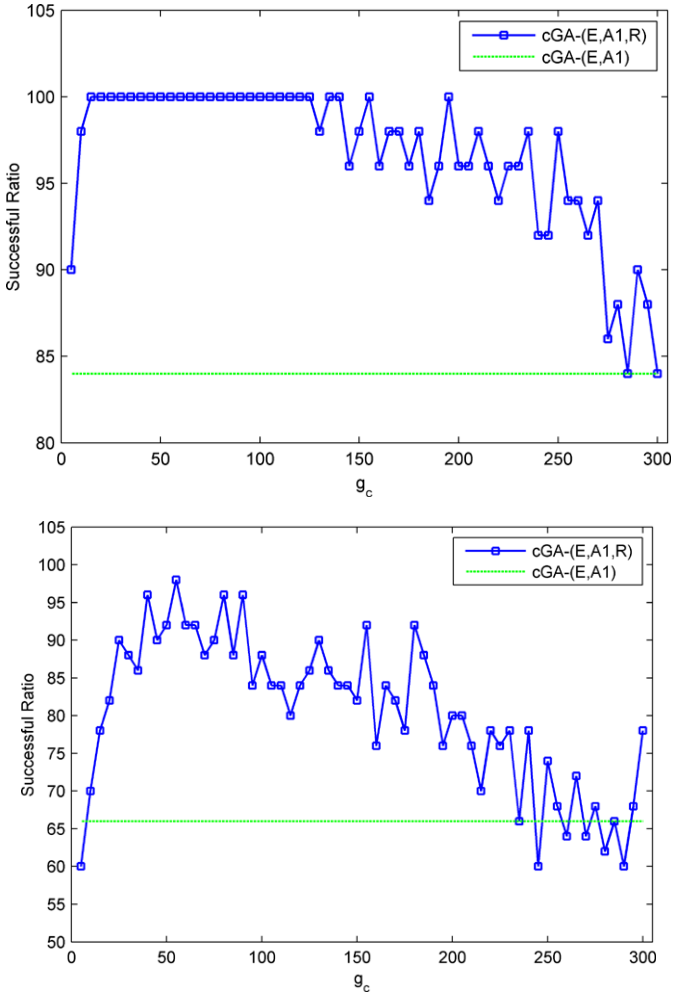

Fig. 10 Successful ratio vs. g_c in variants of cGA. (**a**) The 3-copies network (**b**) the 20-node network

Figure 9 shows the comparisons of the three algorithms in terms of the average fitness during the evolution. Compared with cGA-(E) and cGA-(E, A1), cGA has the worst performance on both networks. In the 3-copies network, cGA has almost no evolution. In the 20-node network, the convergence can hardly be observed before the 200th generation. cGA-(E) performs better than cGA on both networks, which once again demonstrates the effectiveness of the persistent elitist scheme over the traditional tournament scheme [29]. With the use of the all-one vector, cGA-(E, A1) performs the best as shown by the significantly improved convergence speed. For example, cGA-(E, A1) converges to a stable state at around the 150th generation for the 3-copies network and the 200th generation for the 20-node network, respectively. This is because, with more feasible solutions quickly generated from the all-one vector, cGA-(E, A1) is able to converge much faster to better solutions.

Experimental results of different algorithm evaluation criteria for each algorithm are presented in Table 1. Similarly, we found that cGA-(E, A1) has the best performance with the highest successful ratio, and the smallest average best fitness, standard deviation and average termination generation. Besides, the performance of cGA-(E, A1) is significantly improved, i.e. 86% successful ratio by cGA-(E, A1) compared to 24% by cGA-(E) and only 6% by cGA. The obtained results clearly show that the all-one vector does improve the optimization performance of cGA.

5.2 The effect of the PV restart scheme

To illustrate the effectiveness of the PV restart scheme, we compare the following two variants of algorithms on the 3-copies and 20-node networks:

- cGA-(E, A1)
- cGA-(E, A1, R): cGA-(E, A1) with the PV restart scheme.

We investigate the effect of the pre-defined number of consecutive generations in the PV restart scheme, i.e. g_c, upon the performance of the cGA (see Sect. 4.3). The successful ratio and average termination generation of cGA-(E, A1, R) have been obtained with g_c set to 5, 10, 15, ..., 295, and 300, respectively (increment of 5 generations). For com-

Table 1 Experimental results of the three algorithms

Scenarios	Criteria	cGA	cGA-(E)	cGA-(E, A1)
3-copies	s.r. (%)	6	24	86
	a.b.f(s.d.)	40.24(19.72)	28.20(24.84)	0.14(0.35)
	a.t.g.	292.16	264.66	78.78
20-node	s.r.(%)	6	38	78
	a.b.f.(s.d.)	20.82(24.08)	8.58(18.27)	0.22(0.41)
	a.t.g.	283.94	243.20	111.86

Note: s.r.: successful ratio; a.t.g.: average termination generation; a.b.f.: average best fitness; s.d.: standard deviation

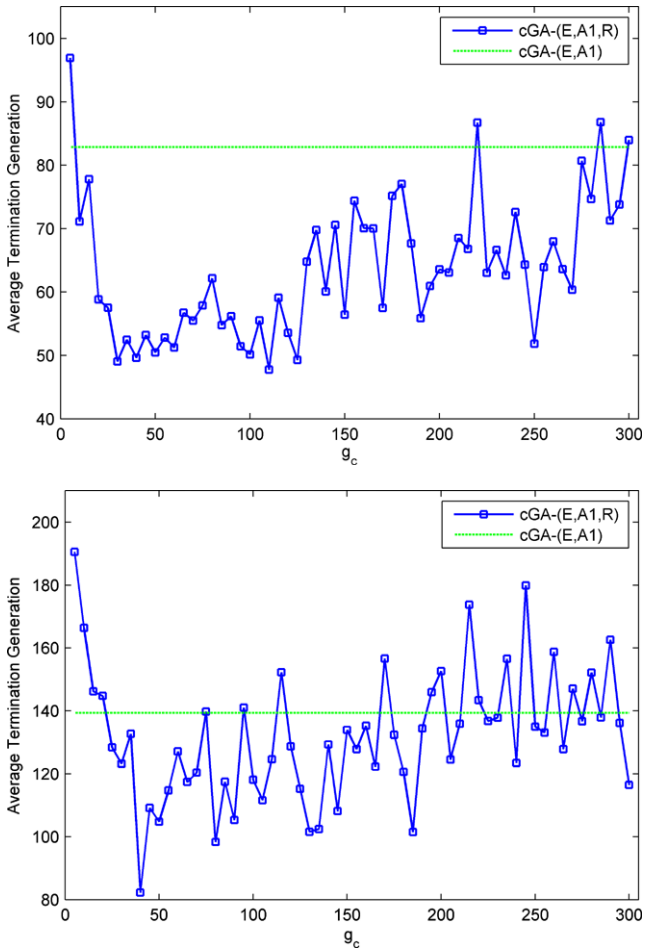

Fig. 11 Average termination generation vs. g_c in variants of cGA. (**a**) The 3-copies network (**b**) the 20-node network

parison purposes, we also collect the successful ratio and average termination generation of cGA-(E, A1), by running it 50 times. The successful ratio and average termination generation are 84% and 82.7 for the 3-copies network, and 66% and 139.3 for the 20-node network, respectively.

Figure 10 shows the successful ratios obtained by cGA-(E, A1, R) with different g_c. With the incensement of g_c, the successful ratio of cGA-(E, A1, R) firstly increases, and then falls down. For both networks, the successful ratios of cGA-(E, A1, R) are better in most cases than that of cGA-(E, A1), e.g. from $g_c = 5$ to $g_c = 250$ on the 3-copies network and from $g_c = 10$ to $g_c = 200$ on the 20-node network. We can conclude that g_c, when properly set, contributes to a better successful ratio of cGA-(E, A1, R).

Figure 11 illustrates the average termination generations achieved on the two networks. We can hardly find a clear relationship between the average termination generations and g_c. However, we can see that the average termination generations in the first half of the range of g_c are more likely to be better than the average termination generations obtained by cGA-(E, A1). Once again, this shows that g_c when properly set helps to improve the optimization performance of cGA-(E, A1, R).

Table 2 Experimental results of the three algorithms

Scenarios	Criteria	cGA-(E, A1)	cGA-(E, A1, R)	cGA-(E, A1, R, L)
3-copies	s.r.(%)	92	100	100
	a.b.f.(s.d.)	0.08 (0.27)	0.00 (0.00)	0.00 (0.00)
	a.t.g.	64.90	53.2	0
20-node	s.r.(%)	70	92	100
	a.b.f.(s.d.)	0.30 (0.46)	0.08 (0.27)	0.00 (0.00)
	a.t.g.	134.16	112.56	23.20

Note: s.r.: successful ratio; a.t.g.: average termination generation; a.b.f.: average best fitness; s.d.: standard deviation

We can also find that with $g_c = 50$, the successful ratios and average termination generations obtained by cGA-(E, A1, R) all perform better than those obtained by cGA-(E, A1). In the following experiments, we set $g_c = 50$ in the PV restart scheme.

5.3 The effect of the local search operator

As analyzed before, the all-one vector and the PV restart scheme both show to be effective for solving the two testing problems, i.e. the 3-copies and 20-node networks. Besides, the PV restart scheme further improves the performance of cGA-(E, A1). In this subsection, we evaluate the effect of the L-operator and verify whether the advantages of the three improvements can be cascaded, by running the following three variants of cGA in the 3-copies and 20-node networks:

- cGA-(E, A1)
- cGA-(E, A1, R)
- cGA-(E, A1, R, L): cGA-(E, A1, R) with the L-operator.

Table 2 shows the experimental results of the three algorithms. We find that cGA-(E, A1, R, L) performs the best, obtaining the highest successful ratio and the smallest average best fitness and average termination generation. The second best algorithm is cGA-(E, A1, R). Clearly, the L-operator improves the performance of cGA-(E, A1, R). In addition, the most significant improvement is the outstandingly reduced average termination generation from cGA-(E, A1, R, L). Note that the average termination generation of cGA-(E, A1, R, L) is zero on the 3-copies network, meaning a coding-free NCM subgraph can be found at the initialization of the algorithm with the L-operator.

5.4 The overall performance analysis

In order to thoroughly analyze the overall performance of the proposed algorithms, we compare the following algorithms in terms of the above same evaluation criteria and the average computational time:

- QEA: the quantum-inspired evolutionary algorithm proposed in [8] for coding resource optimization problem. Based on the standard QEA, this QEA is featured with the multi-granularity evolution mechanism, the adaptive quantum mutation operation and the penalty-function-based fitness function. Besides, it adopts the BLS genotype encoding.
- sGA-1: the simple genetic algorithm with the block transmission state (BTS) encoding and operators presented in [11]. A greedy sweep operator is employed after the evolution to improve the quality of the best individual found.
- sGA-2: the other simple genetic algorithm with BLS encoding and operators in [11]. The same greedy sweep operator is adopted.
- cGA-1: cGA-(E, A1)
- cGA-2: cGA-(E, A1, R)
- cGA-3: cGA-(E, A1, R, L)

Note that the above cGAs are also based on BLS encoding (see Sect. 4.5). The population sizes for the QEA, sGA-1 and sGA-2 are all set to 20. For parameter settings on the calculations of the rotation angle and mutation probabilities in QEA, please refer to [8] for more details. The crossover probability, tournament size and mutation probability are set to 0.8, 12, and 0.012 for sGA-1, and 0.8, 4, and 0.01 for sGA-2, respectively. Experiments have been carried out upon three fixed and eight randomly-generated directed networks. To ensure a fair comparison, QEA, sGA-1 and sGA-2 have been re-implemented on the same machine and evaluated on the same 11 network scenarios. Table 3 shows the experimental networks and parameter setup. All experiments have been run on a Windows XP computer with Intel(R) Core(TM)2 Duo CPU E8400 3.0 GHz, 2GRAM. The results achieved by each algorithm are averaged over 50 runs.

Table 4 compares the six algorithms with respect to the successful ratio. Obviously, cGA-3 is the best algorithm, achieving 100% successful ratio in almost every scenario except Random-5 and Random-6 where cGA-3 also achieves the highest successful ratios, i.e. 98% and 96% respectively. This demonstrates that the three extensions to a large extent strengthen the global exploration and local exploitation of cGA-3. sGA-1 performs the second best. Apart from the Random-5, Random-7 and Random-8 networks, the successful ratios obtained by sGA-1 are at least not worse and usually higher than those obtained by the other four algorithms. Without the local search operator, cGA-2 shows to

Table 3 Experimental networks and parameter setup

Multicast scenario description					Parameters	
name	nodes	links	sinks	rate	LI	DTG
7-copies	57	84	8	2	80	500
15-copies	121	180	16	2	176	500
31-copies	249	372	32	2	368	1000
Random-1	30	60	6	3	86	500
Random-2	30	69	6	3	112	500
Random-3	40	78	9	3	106	500
Random-4	40	85	9	4	64	500
Random-5	50	101	8	3	145	500
Random-6	50	118	10	4	189	500
Random-7	60	150	11	5	235	1000
Random-8	60	156	10	4	297	1000

Note: LI: the length of an individual; DTG: the defined termination generation

Table 4 Comparisons of successful ratio (%)

Scenarios	QEA	sGA-1	sGA-2	cGA-1	cGA-2	cGA-3
7-copies	45	96	80	42	100	100
15-copies	0	50	0	0	4	100
31-copies	0	0	0	0	0	100
Random-1	100	100	98	94	100	100
Random-2	100	100	100	100	100	100
Random-3	66	70	50	24	68	100
Random-4	100	100	100	100	100	100
Random-5	46	40	56	14	26	98
Random-6	42	32	16	18	30	96
Random-7	25	60	6	2	14	100
Random-8	84	60	14	14	80	100

Table 5 Comparisons of average best fitness (standard deviation)

Scenarios	QEA	sGA-1	sGA-2	cGA-1	cGA-2	cGA-3
7-copies	0.95(1.09)	0.04(0.19)	0.70(1.83)	0.82(0.87)	0.00(0.00)	0.00(0.00)
15-copies	10.2(7.09)	0.60(0.68)	4.55(3.85)	5.46(1.85)	2.42(1.23)	0.00(0.00)
31-copies	18.8(5.35)	3.85(1.13)	22.5(6.36)	17.64(2.68)	7.60(2.39)	0.00(0.00)
Random-1	0.00(0.00)	0.00(0.00)	0.02(0.14)	0.06(0.23)	0.00(0.00)	0.00(0.00)
Random-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)
Random-3	0.32(0.47)	0.30(0.47)	0.50(0.51)	1.10(0.76)	0.32(0.47)	0.00(0.00)
Random-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)
Random-5	0.55(0.51)	0.64(0.48)	0.50(0.51)	1.04(0.56)	0.74(0.44)	0.02(0.14)
Random-6	0.60(0.59)	0.94(0.84)	1.15(0.74)	1.34(0.93)	0.94(0.73)	0.04(0.19)
Random-7	1.50(1.23)	0.35(0.48)	1.00(0.32)	2.22(0.95)	1.48(0.93)	0.00(0.00)
Random-8	0.16(0.37)	0.35(0.48)	0.90(0.44)	1.24(0.74)	0.20(0.40)	0.00(0.00)

be weak on local exploitation, however, is still able to obtain decent results compared with sGA-2.

Experimental results of the average best fitness and standard deviation for each algorithm are shown in Table 5. Obviously, cGA-3 outperforms all the other algorithms while cGA-1 and cGA-2 cannot see outstanding advantage when they are compared with QEA, sGA-1 and sGA2. The results also show that the three extensions significantly improve the optimization performance of cGA.

Table 6 illustrates the comparisons of the average termination generation obtained by each algorithm. It is easy to identify that cGA-3 performs outstandingly better than other

Table 6 Comparisons of average termination generation

Scenarios	QEA	sGA-1	sGA-2	cGA-1	cGA-2	cGA-3
7-copies	301.2	228.3	289.6	328.9	233.6	0.0
15-copies	500.0	458.2	500.0	500.0	497.5	0.0
31-copies	1000.0	1000.0	1000.0	1000.0	1000.0	0.0
Random-1	9.7	85.3	66.7	60.5	33.7	0.0
Random-2	6.5	44.7	51.5	34.0	46.2	19.5
Random-3	225.0	338.6	398.1	405.4	309.8	72.2
Random-4	7.4	36.8	32.0	29.0	32.0	0.0
Random-5	349.4	393.3	355.6	443.3	420.6	152.34
Random-6	338.6	436.4	457.7	435.4	425.6	136.3
Random-7	832.2	755.0	989.9	982.5	924.1	183.2
Random-8	300.5	753.1	891.5	875.9	507.4	114.1

Table 7 Comparisons of average computational time (s)

Scenarios	QEA	sGA-1	sGA-2	cGA-1	cGA-2	cGA-3
7-copies	25.15	16.82	14.14	3.06	1.38	0.11
15-copies	195.57	158.24	112.68	23.03	16.28	2.04
31-copies	3903.5	2406.2	436.85	399.05	269.71	28.32
Random-1	0.51	6.22	3.40	0.35	0.10	0.06
Random-2	0.62	3.25	2.26	0.15	0.15	0.16
Random-3	27.97	31.74	31.55	5.17	2.56	3.60
Random-4	0.68	3.37	2.58	0.09	0.11	0.04
Random-5	56.73	57.69	41.72	7.39	4.64	9.33
Random-6	75.20	78.14	63.83	10.77	8.14	16.90
Random-7	292.28	225.32	272.79	43.05	36.61	46.00
Random-8	120.90	229.32	224.23	37.83	14.35	22.83

algorithms, terminating in the initial generation in five networks and in a significantly reduced number of generations in the other networks. As mentioned in Sect. 5.3, cGA-3 also terminates in the initial generation on the 3-copies network. Note that termination in the initial generation occurs only when a coding-free NCM subgraph is found in the initialization of cGA-3. This phenomenon shows that combining the L-operator with the all-one vector is particularly effective to solve n-copies networks and Random-1 and Random-4 network problems.

The computational time is of vital importance in evaluating an algorithm. Table 7 illustrates that cGA-3 spends less time than QEA, sGA-1 and sGA-2 on all networks concerned and sometimes the time reduction can be substantial. For example, in the case of the 31-copies network, the average computational time of cGA-3 is around 30 s, which is significantly shorter than 3993 s by QEA and 2406 s by sGA-1. This demonstrates that, integrated with intelligent schemes, our proposed cGA consumes less computational time while obtaining better solutions compared with the existing algorithms. The reason for cGA-3 consuming less computational time is that the number of fitness evaluations is far reduced in each elitism-based cGA as it only produces one individual at each generation. Although more time may be spent on the local search procedure, the high quality solutions found by the L-operator and the less number of fitness evaluations can well compromise and lead to less computational expenses.

In summary, with regard to the successful ratio, the average best fitness and standard deviation, the average termination generation and the average computational time, cGA with the three improvement schemes outperforms all other algorithms being compared.

6 Conclusion and future work

In this paper, we investigated an elitist compact genetic algorithm (cGA) with three improvements for the coding resource minimization problem in network coding multicast. The first improvement is to set the initial elite individual as an all-one vector in the initialization, which not only makes sure that cGA starts with a feasible individual but also gradually tunes the probability vector (PV) so that feasible individuals appear with increasingly higher probabilities. The

second improvement is to reset the PV during the evolution as a previously recorded value so as to improve the global exploration capability of the cGA. The third one is a local search operator that exploits the neighbors of the network coding based multicast subgraph of each feasible individual to improve the solution quality of the cGA. The three improvements, when employed together, significantly improve the performance of our proposed cGA. Furthermore, the proposed cGA is easy to implement, consumes less computational expenses and memory resources compared with standard evolutionary algorithms. This is important as the lower average computational time by cGA may offer a possibility of applying the proposed algorithm to real-time and dynamic communications networks where computational time is crucial.

Our experiments have demonstrated the efficiency of the PV restart scheme, and showed that the parameter g_c needs to be set properly. In this paper, we empirically determined the fixed values of g_c. In our future work, we will investigate how to extend our algorithm so that it can adaptively determine g_c for different given network problems. Besides, the experimental results also showed that our proposed cGA is more effective on n-copies networks and two of the specific random network problems concerned. This raises an interesting research direction to further investigate features of specific network topologies to improve the local search procedure in our evolutionary algorithms.

Acknowledgement This work was supported in part by China Scholarship Council, China, and The University of Nottingham, UK.

References

1. Ahlswede R, Cai N, Li SYR, Yeung RW (2000) Network information flow. IEEE Trans Inf Theory 46(4):1204–1216
2. Li SYR, Yeung RW, Cai N (2003) Linear network coding. IEEE Trans Inf Theory 49(2):371–381
3. Koetter R, Médard M (2003) An algebraic approach to network coding. IEEE/ACM Trans Netw 11(5):782–795
4. Wu Y, Chou PA, Kung SY (2005) Minimum-energy multicast in mobile ad hoc networks using network coding. IEEE Trans Commun 53(11):1906–1918
5. Chou PA, Wu Y (2007) Network coding for the internet and wireless networks. IEEE Signal Process Mag 24(5):77–85
6. Cai N, Yeung RW (2002) Secure network coding. In: Proceedings of IEEE international symposium on information theory (ISIT'02)
7. Kamal AE (2006) $1+N$ protection in optical mesh networks using network coding on p-cycles. In: Proceedings of IEEE globecom, San Francisco
8. Xing H, Ji Y, Bai L, Sun Y (2010) An improved quantum-inspired evolutionary algorithm for coding resource optimization based network coding multicast scheme. AEÜ, Int J Electron Commun 64(12):1105–1113
9. Kim M, Ahn CW, Médard M, 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, 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, Kim W (2007) Genetic representations for evolutionary optimization of network coding. In: Proceedings of evoworkshops 2007. LNCS, Valencia, vol 448, pp 21–31
12. Langberg M, Sprintson A, Bruck J (2006) The encoding complexity of network coding. IEEE Trans Inf Theory 52(6):2386–2397
13. Oliveira CAS, Pardalos PM (2005) A Survey of Combinatorial Optimization Problems in Multicast Routing. Comput Oper Res 32(8):1953–1981
14. Yeo CK, Lee BS, Er MH (2004) A survey of application level multicast techniques. Comput Commun 27:1547–1568
15. Xu Y, Qu R (2010) A hybrid scatter search meta-heuristic for delay-constrained multicast routing problems. Appl Intell. doi:10.1007/s10489-010-0256-x
16. Araújo AFR, Garrozi C (2010) MulRoGA: a multicast routing genetic algorithm approach considering multiple objectives. Appl Intell 32:330–345. doi:10.1007/s10489-008-0148-5
17. Kim SJ, Choi MK (2007) Evolutionary algorithms for route selection and rate allocation in multirate multicast networks. Appl Intell 27:197–215 doi:10.1007/s10489-006-0014-2
18. Fragouli C, Soljanin E (2006) Information flow decomposition for network coding. IEEE Trans Inf Theory 52(3):829–848
19. Xing H, Qu R (2011) A population based incremental learning for delay constrained network coding resource minimization. In: Proceedings of evoapplications 2011, Torino, Italy, pp 51–60
20. Pelikan M, Goldberg DE, Lobo FG (2002) A survey of optimization by building and using probabilistic models. Comput Optim Appl 21:5–20
21. Baluja S, Simon D (1998) Evolution-based methods for selecting point data for object localization: applications to computer-assisted surgery. Appl Intell 8:7–19
22. Sukthankar R, Baluja S, Hancock J (1998) Multiple adaptive agents for tactical driving. Appl Intell 9:7–23
23. Harik GR, Lobo FG, Goldberg DE (1999) The compact genetic algorithm. IEEE Trans Evol Comput 3(4):287–297
24. Gallagher JC, Vigraham S, Kramer G (2004) A family of compact genetic algorithms for intrinsic evolvable hardware. IEEE Trans Evol Comput 8(2):111–126
25. Aporntewan C, Chongstitvatana P (2001) A hardware implementation of the compact genetic algorithm. In: Proceedings of IEEE congress evolutionary computation. pp 624–629
26. JI Hidalgo, Baraglia R, Perego R, Lanchares J, Tirado F (2001) A parallel compact genetic algorithm for multi-FPGA partitioning. In: Proceedings of the 9th workshop on parallel and distributed processing, Mantova. pp 113–120
27. Silva RR, Lopes HS, Erig Lima CR (2008) A compact genetic algorithm with elitism and mutation applied to image recognition. In: Proceedings of the 4th international conference on intelligent computing (ICIC'08). pp 1109–1116
28. Lin SF, Chang JW, Hsu YC (2010) A self-organization mining based hybrid evolution learning for TSK-type fuzzy model design. Appl Intell. doi:10.1007/s10489-010-0271-y
29. Ahn CW, Ramakrishna RS (2003) Elitism-based compact genetic algorithm. IEEE Trans Evol Comput 7(4):367–385
30. Goldberg AV (1985) A new max-flow algorithm. MIT Technical report MIT/LCS/TM-291, Laboratory for Computer Science
31. Yang S, Yao X (2008) Population-based incremental learning with associative memory for dynamic environments. IEEE Trans Evol Comput 12(5):542–561
32. Yang S, Yao X (2005) Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Comput 9(11):815–834