Genetic Algorithms With Immigrants and Memory Schemes for Dynamic Shortest Path Routing Problems in Mobile *Ad Hoc* Networks

Shengxiang Yang, *Member, IEEE*, Hui Cheng, and Fang Wang, *Member, IEEE*

***Abstract*—In recent years, the static shortest path (SP) problem has been well addressed using intelligent optimization techniques, e.g., artificial neural networks, genetic algorithms (GAs), particle swarm optimization, etc. However, with the advancement in wireless communications, more and more mobile wireless networks appear, e.g., mobile networks [mobile *ad hoc* networks (MANETs)], wireless sensor networks, etc. One of the most important characteristics in mobile wireless networks is the topology dynamics, i.e., the network topology changes over time due to energy conservation or node mobility. Therefore, the SP routing problem in MANETs turns out to be a dynamic optimization problem. In this paper, we propose to use GAs with immigrants and memory schemes to solve the dynamic SP routing problem in MANETs. We consider MANETs as target systems because they represent new-generation wireless networks. The experimental results show that these immigrants and memory-based GAs can quickly adapt to environmental changes (i.e., the network topology changes) and produce high-quality solutions after each change.**

***Index Terms*—Dynamic optimization problem (DOP), dynamic shortest path routing problem (DSPRP), genetic algorithm (GA), immigrants scheme, memory scheme, mobile *ad hoc* network (MANET).**

I. Introduction

MOBILE *ad hoc* network (MANET) [25], [26], [28] is a self-organizing and self-configuring multihop wireless network, which is composed of a set of mobile hosts (MHs) that can move around freely and cooperate in relaying packets on behalf of one another. MANET supports robust and efficient operations by incorporating the routing functionality into MHs. In MANETs, the unicast routing establishes a multihop forwarding path for two nodes beyond the direct wireless communication range. Routing protocols also maintain connectivity when links on these paths break due to effects such as node movement, battery drainage, radio propagation, and wireless interference. In multihop networks, routing is one of the most important issues that has a significant impact on the performance of networks. So far, there are mainly two types of routing protocols in MANETs, namely, topological routing and geographic routing. In the topological routing, mobile nodes utilize the topological information to construct routing tables or search routes directly. In the geographic routing, each node knows its own position and makes routing decisions based on the position of the destination and the positions of its local neighbors.

In this paper, we investigate the shortest path (SP) routing problem, which belongs to the topological routing. The SP routing problem aims to find the SP from a specific source to a specific destination in a given network while minimizing the total cost associated with the path. The SP routing problem is a classical combinatorial optimization problem arising in many design and planning contexts [1], [3]. There are several deterministic search algorithms for the SP problem: the Dijkstra's algorithm, the breadth-first search algorithm, the Bellman–Ford algorithm, etc. All these algorithms have a polynomial time complexity. They are effective in fixed infrastructure wireless or wired networks. But, they exhibit an unacceptable high computational complexity for real-time communications involving rapidly changing network topologies [2]. Therefore, for the dynamic SP routing problem (DSPRP) in a changing network environment, we need to employ appropriate new approaches. The DSPRP has become a topic of interest in recent years, and there are some works dealing with DSPRPs in the literature [10], [16].

The DSPRP in MANETs is a real-world dynamic optimization problem (DOP). In recent years, studying evolutionary algorithms (EAs) for DOPs has attracted a growing interest due to its importance in EA's real-world applications [43]. The simplest way of addressing DOPs is to restart EAs from scratch whenever an environment change is detected. Though the restart scheme really works for some cases [42], for many DOPs, it is more efficient to develop other approaches that make use of knowledge gathered from old environments. Over the years, several approaches have been developed for EAs to address dynamic environments [5], [19], [38], such as maintaining diversity during the run via random immigrants [12], [32], increasing diversity after a change [8], using memory schemes to reuse stored useful information [4], [29], [30], [34], [35], applying multipopulation and speciation schemes to search in different regions of the search space [6], [22], [23], and adapting (the parameters of) operators to quickly respond to a new environment [20], [39], [41].

In this paper, we adapt and investigate several genetic algorithms (GAs) that are developed to deal with general DOPs to solve the DSPRP in MANETs. First, we design the components of the standard GA (SGA) specifically for the DSPRP. Then, we integrate several immigrants and memory schemes and their combination into the GA to enhance its searching

Manuscript received February 27, 2009. First published July 7, 2009; current version published November 2, 2009. This work was supported by the Engineering and Physical Sciences Research Council of U.K. under Grant EP/E060722/1. This paper was recommended by Associate Editor J. Lazansky.

S. Yang and H. Cheng are with the Department of Computer Science, University of Leicester, Leicester LE1 7RH, U.K. (e-mail: s.yang@mcs.le.ac.uk; hc118@mcs.le.ac.uk).

F. Wang is with the Centre for Information and Security Systems Research, British Telecommunications Innovate, Ipswich IP5 3RE, U.K. (e-mail: fang.wang@bt.com.)

Digital Object Identifier 10.1109/TSMCC.2009.2023676

capacity for the SPs in dynamic environments. Once the topology is changed, new immigrants or the useful information stored in the memory can help guide the search of good solutions in the new environment. For comparison purposes, we also implement two traditional GA schemes, i.e., SGA and restart GA, as the peer algorithms. Via simulation experiments, we evaluate these GAs under different parameter settings to find the best combinations. More importantly, we evaluate them under various settings of dynamic environments to see their performance and find the best match between algorithms and environmental characteristics. Generally speaking, the investigated well-designed GAs work well for the DSPRP in MANETs.

The rest of this paper is organized as follows. We discuss related work in Section II. The MANET network model and the DSPRP model are described in Section III. Section IV presents the design of a specialized GA for the static SP routing problem. The GAs that are the integration of several immigrants and/or memory schemes and the specialized GA for the DSPRP are described in Section V. The extensive experimental study and relevant analysis are presented in Section VI. Finally, Section VII concludes this paper with some discussions on the future work.

II. Related Work

The SP problem has been investigated extensively in the literature. Since deterministic algorithms with a polynomial time complexity are not suitable for the real-time computation of SPs, quite a few research works have been conducted to solve the SP problem using artificial intelligence techniques, e.g., artificial neural networks [1], GAs [2], and particle swarm optimization (PSO) [17].

In [1], a near-optimal routing algorithm employing a modified Hopfield neural network (HNN) was proposed. It uses every piece of information that is available at the peripheral neurons, in addition to the highly correlated information that is available at the local neuron. Therefore, it can achieve a faster convergence and a better route optimality than other HNN-based algorithms. In [2], a genetic algorithmic approach was presented to the SP routing problem. Computer simulations showed that the GA-based SP algorithm exhibits a much better quality of solution (i.e., the route optimality) and a much higher rate of convergence than other algorithms. A population-sizing equation that facilitates a solution with the desired quality was also developed. In [17], a PSO-based search algorithm was proposed. A priority-based indirect path-encoding scheme is used to widen the scope of the search space, and a heuristic operator is used to reduce the probability of invalid loop creations during the path-construction procedure. It was claimed that the PSO-based SP algorithm is superior to those using GAs, including the one in [2].

However, all these algorithms address only the static SP problem. When the network topology changes, they will regard it as a new network and restart the algorithms over the new topology. It is well known that the topology changes rapidly in MANETs due to the characteristics of wireless networks, e.g., battery exhaustion and node mobility. Therefore, for the dynamic SP problem in MANETs, these algorithms are not good choices since, in this regard, immigrants and memory-enhanced GAs (MEGAs) have their inherent advantages. These GAs use the immigrants or the useful information stored in the memory to help the population quickly adapt to the new environment after a change occurs. Hence, these algorithms can keep running over the continuously changing topologies, and avoid the expensive and inefficient restart. To our best knowledge, these dynamic GAs have not been applied to date to solve DOPs in the real-world networks.

III. Dynamic SP Routing Problem

In this section, we first present our network model and then formulate the DSPRP. We consider a MANET operating within a fixed geographical region. We model it by an undirected and connected topology graph $G_0(V_0, E_0)$, where V_0 represents the set of wireless nodes (i.e., routers) and E_0 represents the set of communication links connecting two neighboring routers falling into the radio transmission range. A communication link (i, j) cannot be used for packet transmission unless both node i and node j have a radio interface each with a common channel. However, the channel assignment is beyond the scope of this paper. In addition, message transmission on a wireless communication link will incur remarkable delay and cost.

Here, we summarize some notations that we use throughout this paper:

$G_0(V_0, E_0)$	initial MANET topology graph;
$G_i(V_i, E_i)$	MANET topology graph after the ith change;
s	source node;
r	destination node;
$P_i(s, r)$	path from s to r on the graph G_i;
d_l	transmission delay on the communication link l;
c_l	cost on the communication link l;
$\Delta(P_i)$	total transmission delay on the path P_i;
$C(P_i)$	the total cost of the path P_i.

The DSPRP can be informally described as follows. Initially, given a network of wireless routers, a delay upper bound, a source node, and a destination node, we wish to find a delay-bounded least cost loop-free path on the topology graph.[1] Then, periodically or stochastically, due to energy conservation or some other issues, some nodes are scheduled to sleep or some sleeping nodes are scheduled to wake up. Therefore, the network topology changes from time to time. The objective of the DSPRP is to quickly find the new optimal delay-constrained least cost acyclic path after each topology change.

More formally, consider MANET $G(V, E)$ and a unicast communication request from the source node s to the destination node r with the delay upper bound Δ. The *dynamic delay-constrained SP problem* is to find a series of paths $\{P_i | i \in \{0, 1, \ldots\}\}$ over a series of graphs $\{G_i | i \in \{0, 1, \ldots\}\}$, which satisfy the delay constraint, as shown in (1), and have the

[1] Since the end-to-end delay [24] is a pretty important quality-of-service (QoS) metric to guarantee the real-time data delivery, we also require the routing path to satisfy the delay constraint.

least path cost, as shown in (2):

$$\Delta(P_i) = \sum_{l \in P_i(s,r)} d_l \leq \Delta \tag{1}$$

$$C(P_i) = \min_{P \in G_i} \left\{ \sum_{l \in P(s,r)} c_l \right\}. \tag{2}$$

IV. Specialized GA for the SP Problem

This section describes the design of the GA for the SP problem. The design of the GA involves several key components: genetic representation, population initialization, fitness function, selection scheme, crossover, and mutation. A routing path consists of a sequence of adjacent nodes in the network. Hence, it is a natural choice to adopt the path-oriented encoding method. For the routing problem, the path-oriented encoding and the path-based crossover and mutation are also very popular [2], [11].

A. Genetic Representation

A routing path is encoded by a string of positive integers that represent the IDs of nodes through which the path passes. Each locus of the string represents an order of a node (indicated by the gene of the locus). The gene of the first locus is for the source node and the gene of the last locus is for the destination node. The length of a routing path should not exceed the maximum length $|V_0|$, where V_0 is the set of nodes in the MANET. Chromosomes are encoded under the delay constraint. In case it is violated, the encoding process is usually repeated so as to satisfy the delay constraint.

B. Population Initialization

In the GA, each chromosome corresponds to a potential solution. The initial population Q is composed of a certain number of, say q, chromosomes. To promote the genetic diversity, in our algorithm, the corresponding routing path is randomly generated for each chromosome in the initial population. We start to search a random path from s to r by randomly selecting a node v_1 from $N(s)$, the neighborhood of s. Then, we randomly select a node v_2 from $N(v_1)$. This process is repeated until r is reached. Since the path should be loop-free, those nodes that are already included in the current path are excluded from being selected as the next node to be added into the path, thereby avoiding reentry of the same node into a path. In this way, we get a random path $P(s, r) = \{s, v_1, v_2, \ldots, r\}$. Repeating this process for q times, the initial population $Q = \{\mathrm{Ch}_0, \mathrm{Ch}_1, \ldots, \mathrm{Ch}_{q-1}\}$ can be obtained.

C. Fitness Function

Given a solution, we should accurately evaluate its quality (i.e., the fitness value), which is determined by the fitness function. In our algorithm, we aim to find the least cost path between the source and the destination. Our primary criterion of solution quality is the path cost. Therefore, among a set of candidate solutions (i.e., unicast paths), we choose the one with the least path cost. The fitness value of chromosome Ch_i (representing the path P), denoted as $f(\mathrm{Ch}_i)$, is given by

$$f(\mathrm{Ch}_i) = \left[\sum_{l \in P(s,r)} c_l \right]^{-1}. \tag{3}$$

The proposed fitness function is to be maximized and involves only the total path cost. As mentioned earlier, the delay constraint is checked for each chromosome during the evolutionary process.

D. Selection Scheme

Selection plays an important role in improving the average quality of the population by passing the high-quality chromosomes to the next generation. The selection of chromosome is based on the fitness value. We adopt the scheme of pairwise tournament selection without replacement [13] as it is simple and effective. The tournament size is 2.

E. Crossover and Mutation

A GA relies on two basic genetic operators—crossover and mutation. Crossover processes the current solutions so as to find better ones. Mutation helps a GA keep away from local optima [2]. The performance of a GA very much depends on them. The type and implementation of operators depend on encoding as well as the problem in hand.

In our algorithm, since chromosomes are expressed by the path structure, we adopt the single-point crossover to exchange partial chromosomes (subpaths) at positionally independent crossing sites between two chromosomes [2]. With the crossover probability, each time we select two chromosomes Ch_i and Ch_j for crossover. Ch_i and Ch_j should possess at least one common node. Among all the common nodes, one node, denoted as v, is randomly selected. In Ch_i, there is a path consisting of two parts: $(s \xrightarrow{\mathrm{Ch}_i} v)$ and $(v \xrightarrow{\mathrm{Ch}_i} r)$. In Ch_j, there is a path consisting of two parts: $(s \xrightarrow{\mathrm{Ch}_j} v)$ and $(v \xrightarrow{\mathrm{Ch}_j} r)$. The crossover operation exchanges the subpaths $(v \xrightarrow{\mathrm{Ch}_i} r)$ and $(v \xrightarrow{\mathrm{Ch}_j} r)$.

The population will undergo the mutation operation after the crossover operation is performed. With the mutation probability, each time we select one chromosome Ch_i on which one gene is randomly selected as the mutation point (i.e., mutation node), denoted as v. The mutation will replace the subpath $(v \xrightarrow{\mathrm{Ch}_i} r)$ by a new random subpath.

Both crossover and mutation may produce new chromosomes that represent infeasible solutions. Therefore, we check if the path represented by a new chromosome is acyclic. If not, a repair function [21] will be applied to eliminate the loops. The delay checking is incorporated into the crossover and mutation operations to guarantee that all new chromosomes produced by crossover or mutation satisfy the delay constraint.

V. Investigated GAs for the DSPRP

A. Traditional GAs

For the DSPRP, we can still address them using the specialized GA described earlier with two variants, denoted as *SGA* and *restart GA*. In the SGA, when an environmental change leads to infeasible solutions, SGA handles them by taking the measure of penalty, i.e., infeasible solutions are set to a very low fitness. In this way, the population in SGA can keep evolving even in a continuously changing environment. In the restart GA, once a change is detected, the population will be reinitialized based on the new network topology.

B. GAs With Immigrants Schemes

In stationary environments, convergence at a proper pace is really what we expect for GAs to locate the optimum solutions for many optimization problems. However, for DOPs, convergence usually becomes a big problem for GAs because changing environments usually require GAs to keep a certain population diversity level to maintain their adaptability. To address this problem, the random immigrants approach is a quite natural and simple way [12], [27], [40], [44], [45]. The random immigrants based GA (RIGA) was proposed by Grefenstette with the inspiration from the flux of immigrants that wander in and out of a population between two generations in nature. It maintains the diversity level of the population through replacing some individuals of the current population with random individuals, called *random immigrants*, every generation. As to which individuals in the population should be replaced, usually there are two strategies: replacing random individuals or replacing the worst ones [32]. In order that random immigrants do not disrupt the ongoing search progress too much, especially during the period when the environment does not change, the ratio of the number of random immigrants to the population size is usually set to a small value, e.g., 0.2.

However, in a slowly changing environment, the introduced random immigrants may divert the searching force of the GA during each environment before a change occurs, and hence, may degrade the performance. On the other hand, if the environment changes only slightly in terms of severity of changes, random immigrants may not have any actual effect even when a change occurs because individuals in the previous environment may still be quite fit in the new environment. Based on the previous consideration, an immigrants approach, called elitism-based immigrants, was proposed for GAs to address DOPs [36]. The elitism-based immigrants GA (EIGA) is also investigated for the DSPRP in this paper.

Within the EIGA, for each generation *t*, after the normal genetic operations (i.e., selection and recombination), the elite $E(t-1)$ from the previous generation is used as the base to create immigrants. From $E(t-1)$, a set of $r_{\mathrm{ei}}n$ individuals are iteratively generated by mutating $E(t-1)$ with a probability p_m^i, where n is the population size and r_{ei} is the ratio of the number of elitism-based immigrants to the population size. The generated individuals then act as immigrants and replace the worst individuals in the current population. It can be seen that the elitism-based immigrants scheme combines the idea of elitism with the traditional random immigrants scheme. It uses the elite from the previous population to guide the immigrants toward the current environment, which is expected to improve the performance of GAs in dynamic environments.

```
t := 0 and initialize population $P(0)$ randomly
repeat
  evaluate population $P(t)$
  $P't$ = selectForReproduction($P(t)$)
  crossover($P'(t), p_c$) // $p_c$ is the crossover probability
  mutate($P'(t), p_m$) // $p_m$ is the mutation probability
  evaluate the interim population $P'(t)$

  // perform elitism-based immigration
  denote the elite in $P(t-1)$ by $E(t-1)$
  generate $r_{ei}*n$ immigrants by mutating $E(t-1)$
    with $p_m^i$
  evaluate these elitism-based immigrants

  if the hybrid scheme is used then
    generate $r_{ri}*n$ random immigrants
    evaluate these random immigrants

  replace the worst individuals in $P'(t)$ with the
    generated immigrants
  $P(t+1) := P'(t)$
until the termination condition is met // e.g., $t > t_{max}$
```

Fig. 1. Pseudocode for the EIGA and the HIGA, where the elitism of size one is used.

In order to address significant changes that a DSPRP may suffer, the elitism-based immigrants can be hybridized with the traditional random immigrants scheme. The pseudocode for the GA with the hybrid immigrants scheme, denoted as *hybrid immigrants GA (HIGA)*, is also investigated in this paper. Within HIGA, in addition to the $r_{\mathrm{ei}}n$ immigrants created from the elite of the previous generation, $r_{\mathrm{ri}}n$ immigrants are also randomly created, where r_{ri} is the ratio of the number of random immigrants to the population size. These two sets of immigrants will then replace the worst individuals in the current population.

Fig. 1 shows the pseudocode for EIGA and HIGA. In our implementation of EIGA, if the mutation probability p_m^i is satisfied, the elite $E(t-1)$ will be used to generate the new immigrants by a mutation operation; otherwise, $E(t-1)$ itself will be directly used as a new immigrant.

C. GAs With Memory Schemes

While the immigrants schemes use random immigrants or elitism-based immigrants to maintain the population diversity to adapt to the changing environments, the memory scheme aims to enhance the performance of GAs for DOPs in a different way. It works by storing useful information from the current environment, either implicitly through redundant representations [9], [14], [31] or explicitly by storing good (usually best) solutions of the current population in an extra memory [4], [15], [18]. The stored information can be reused later in new environments. For example, for the explicit memory scheme (which is the concern of this paper), when the environment changes, old solutions in the memory that fit the new environment well will be reactivated, and hence, may adapt GAs to the new environment more

```
$t := 0$ and $t_M := rand(5, 10)$
initialize population $P(0)$ and memory $M(0)$ randomly
repeat
  evaluate population $P(t)$ and memory $M(t)$
  replace the worst individual in $P(t)$ by the elite $E(t-1)$
    from $P(t-1)$
  if change detected then
    $P'(t)$ :=retrieveBestMembersFrom($P(t), M(t)$)
  else $P'(t) := P(t)$
  // time to update memory
  if $t = t_M$ || change detected then
    if $t = t_M$ then
      $B_P(t)$ := retrieveBestMemberFrom($P'(t)$)
    if change detected then $B_P(t) := E(t-1)$
    if still any random point in memory then
      replace a random point in memory with $B_P(t)$
    else    // replace the most similar memory point
      if $t = t_M$ then
        find the memory point $C_M(t)$ closest to $B_P(t)$
        if $f(B_P(t)) > f(C_M(t))$ then
          $C_M(t) := B_P(t)$
      if change detected then
        find the memory point $C_M(t-1)$ closest to $B_P(t)$
        if $f(B_P(t)) > f(C_M(t-1))$ then
          $C_M(t-1) := B_P(t)$
    $t_M := t + rand(5, 10)$
  // standard genetic operations
  $P''(t)$ := selectForReproduction($P'(t)$)
  crossover($P''(t), p_c$) // $p_c$ is the crossover probability
  mutate($P''(t), p_m$)  // $p_m$ is the mutation probability
  $P(t+1) := P''(t)$
until the termination condition is met   // e.g., $t > t_{max}$
```

Fig. 2. Pseudocode for the MEGA.

directly than random immigrants would do. Especially, when the environment changes cyclically, memory can work very well. This is because in cyclic dynamic environments, as time passes, the environment will return to some old environment precisely, and the solution in the memory, which has been optimized with respect to the old environment, will instantaneously move the GA to the reappeared optimum of that environment.

The GA with the memory scheme studied in this paper, called *MEGA* [37], is shown in Fig. 2, where $f(\cdot)$ is the fitness function. MEGA (and other memory-based GAs used in this paper) uses a memory of size $m = 0.1n$. The memory in MEGA is reevaluated every generation to detect environmental changes. The environment is detected as changed if the fitness of at least one individual in the memory is detected to have changed its fitness. If an environmental change is detected, the memory is merged with the current population and the best $n - m$ individuals are selected as an interim population to undergo genetic operations for a new population while the memory remains unchanged.

The memory in MEGA is randomly initialized and also updated in a stochastic time pattern as follows. After each memory updating, a random integer in $[5, 10]$ is generated to decide the next memory updating time t_M. For example, suppose a memory updating happens at generation t, then the next memory updating time is $t_M = t + \mathrm{rand}(5, 10)$. In order to store the most relevant information to an environment in the memory, each time an environmental change is detected, the memory is also updated according to the population just before the environmental change. When the memory is due to update, if any of the randomly initialized points still exists in the memory, the best individual of the current population (if the memory update is due to $t = t_M$) or the elite from the previous population (if the memory update is because an environmental change is detected) will replace one of them randomly; otherwise, the best individual or the elite will replace the closest memory point if it is fitter according to the current environment or the previous environment, respectively.

D. GAs With Memory and Immigrants Schemes

As discussed in the previous section, the random immigrants approach aims to improve GA's performance in dynamic environments through maintaining the population diversity level with random immigrants, and the memory approach aims to move the GA directly to an old environment that is similar to the new one through reusing old good solutions. It is straightforward that the random immigrants and memory approaches can be combined into GAs to deal with DOPs [29]. Therefore, the GA with memory and random immigrants (denoted MRIGA) was developed in [37]. MRIGA differs from MEGA only in that in MRIGA, before entering the next generation, $r_i n$ random immigrants are swapped into the population to replace those worst individuals in the population.

However, a more efficient approach of hybridizing memory and random immigrants for GAs to deal with dynamic environments is the memory-based immigrants scheme, which was proposed in [34] and results in a memory-based immigrants GA (MIGA). MIGA uses the same memory updating scheme as MEGA and MRIGA. However, the memory retrieval does not depend on the detection of environmental changes and is hybridized with the random immigrants scheme via the mutation mechanism. For each generation, the memory is reevaluated, and the best memory point is retrieved as the base to create immigrants. A set of $r_{\mathrm{ri}} n$ individuals are iteratively generated by performing mutation with a probability p_m^i on the best memory point. The generated individuals then act as immigrants and replace the worst $r_{\mathrm{ri}} n$ individuals in the population. In summary, the key idea behind MIGA is that the memory is used to guide the immigrants to make them more biased to the current environment (be it a new one or not) than random immigrants.

VI. Experimental Study

In the simulation experiments, we implement the two traditional GAs (i.e., SGA and restart GA) and the six immigrants and memory-based GAs (i.e., RIGA, EIGA, HIGA, MEGA, MRIGA, and MIGA) for the DSPRP. In SGA, if the change makes one individual in the current population become infeasible (e.g., one or more links in the corresponding path are lost), we add a penalty value to that individual. By simulation experiments, we evaluate their performance in a continuously changing wireless network.

A. Dynamic Test Environments

The initial network topology is generated using the following method. We first specify a square region with the area of 200×200 that has the width [0, 200] on the x-axis and the height [0, 200] on the y-axis. Then we generate 100 nodes, and the position (x, y) of each node is randomly specified within the square area. If the distance between two nodes falls into the radio transmission range D, a link will be added to connect them, and both the cost and the delay of this link are randomly assigned within the corresponding ranges. Finally, we check if the generated topology is connected. If not, the previous process is repeated until a connected topology is generated. In the experiments, D is given a reasonable value of 50.

All the algorithms start from the initial network topology. Then, after a certain number (say, R) of generations (i.e., the change interval), a certain number (say, M) of nodes are scheduled to sleep or wake up depending on their current status. It means that the selected working nodes will be turned off to sleep and the selected sleeping nodes will be turned on to work. Therefore, the network topology is changed accordingly since some links are lost and some other links appear again. By this means, we create a series of network topologies corresponding to the continuous network changes. Furthermore, these adjacent topologies are highly related since each time the changes affect only part of the nodes. It can be seen that R and M determine the change frequency and severity, respectively. The larger the value of R, the slower the changes. The larger the value of M, the more severe the changes.

In the following experiments, we set R to 5, 10, and 15, respectively, to see the impact of the change frequency on the performance of GAs. We also set M to 2, 3, and 4, respectively. Thus, by the number of nodes changed per time, we have three different series of topologies. When M is set to 2, 3, and 4, we generate the topology series #2, #3, and #4, respectively. Each of these three series has 21 different topologies. In addition, since memory schemes are claimed to work well in the cyclicly changing environment, we set M to 2 and generate a cyclic topology series, named as series #1. In topology series #1, topology 1 is the same as topology 21, and the subseries from topology 1 to 21 is repeated five times. Therefore, the cyclic topology series consists of 101 topologies in total. All the experiments are based on the four topology series.

As described in Section IV-D, the GA adopts a pairwise tournament selection without replacement. In all the experiments, the mutation probability is set to 0.1. For RIGA and EIGA, the ratios of the number of immigrants to the population size, $r_{\rm ri}$ and $r_{\rm ei}$, are set to 0.2. However, in HIGA, to guarantee the comparison fairness, i.e., the same number of immigrants are introduced every generation, $r_{\rm ri}$ and $r_{\rm ei}$ are set to 0.1. In EIGA, HIGA, and MIGA, the mutation probability p_m^i for generating new immigrants is set to 0.8. Both the source and destination nodes are randomly selected, and they are not allowed to be scheduled in any change. The delay upper bound Δ is set twice the minimum end-to-end delay.

At each generation, for each algorithm, we select the best individual from the current population and output the cost of the SP represented by it. We first set up basic experiments to evaluate the population size, the impact of the change interval and the change severity, and the improvements over traditional GAs using RIGA, EIGA, and HIGA. Then, since the memory-related schemes (i.e., MEGA, MRIGA, and MIGA) are mainly designed for dynamic environments where changes occur in a cyclic way, we set up cyclic environments to evaluate their performance.

For each experiment of an algorithm on a dynamic problem, ten independent runs are executed with the same set of random seeds. For each run, 21 environmental changes in acyclic dynamic environments and 101 environmental changes in cyclic dynamic environments are allowed. In acyclic dynamic environments, they are equivalent to 105, 210, and 315 generations for $R = 5$, 10, and 15, respectively. In cyclic dynamic environments, they are equivalent to 505, 1010, and 1515 generations for $R = 5$, 10, and 15, respectively. For each run, the best-of-generation fitness is recorded every generation. The overall offline performance of a GA on a DOP is defined as

$$\overline{F}_{\rm BOG} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N} F_{{\rm BOG}_{ij}}\right) \qquad (4)$$

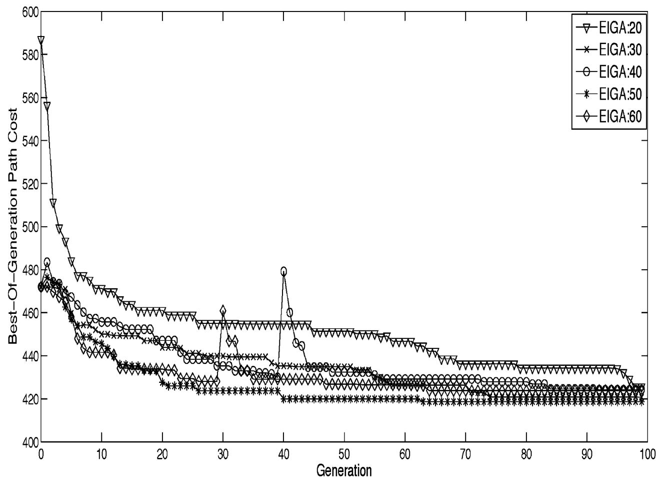
where G is the total number of generations for a run, $N = 10$ is the total number of runs, and $F_{{\rm BOG}_{ij}}$ is the best-of-generation fitness of generation i of run j. $\overline{F}_{\rm BOG}$ is the offline performance, i.e., the best-of-generation fitness averaged over the ten runs and then over the data-gathering period.

B. Basic Experimental Results and Analysis

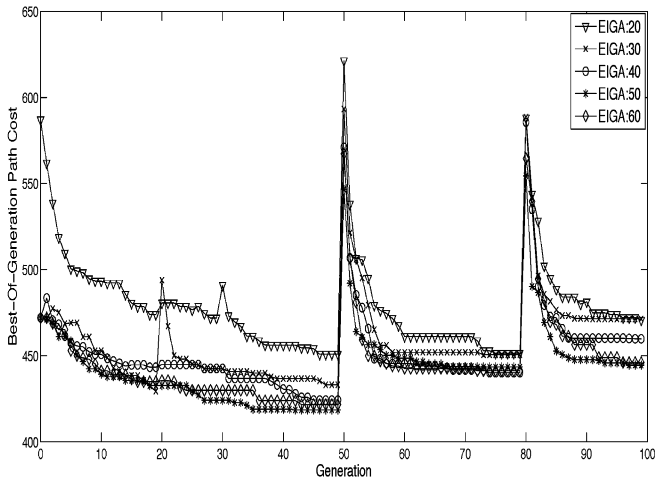
First, we investigate the population size that ensures a specified quality of solution. We pick up EIGA as an example and run it over topology series #2 and #4, respectively. Here, R is set to 10. We vary the population size from 20 to 60 to see if an adequate population size can be determined for this problem. Since there are 21 topologies in each series, EIGA evolves $21R$ generations in total. We sample the first 100 generations to plot the figures. Fig. 3(a) and (b) shows the results over topology series #2 and #4, respectively.

From Fig. 3(a) and (b), it can be seen that the algorithm shows the best performance at the population size of 50. When the population size is increased to 60, the algorithm performance degrades. We check other algorithms and find similar results. Therefore, we conclude that 50 is the best choice for the population size in our problem. From Fig. 3(a) and (b), it can also be seen that sometimes when a change occurs, the algorithms are not affected. The reason is that the topology changes may not always affect the current population, especially the optimal individual in the population. For example, if the nodes that are scheduled to sleep or wake up in one change are not on the path represented by the optimal individual in the population, the optimal individual has a very high probability to stay in the unaffected population. This explains why the algorithms do not always react to the change drastically.

Second, we investigate the impact of the change interval on the algorithm performance. When the change interval is 5, the

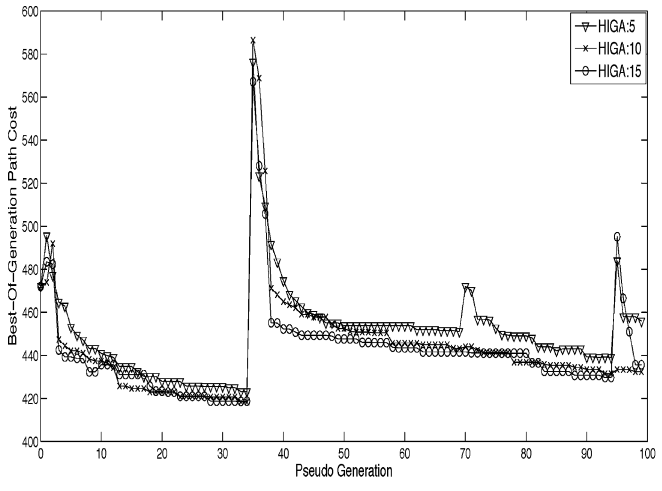


(a)

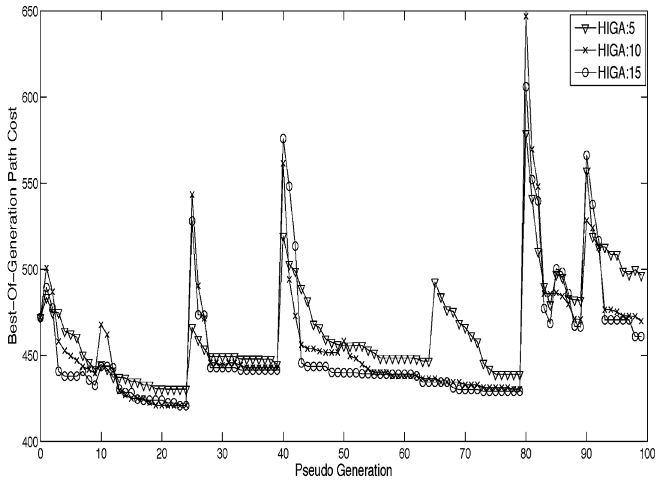


(b)

Fig. 3. Comparison results of the quality of solution for EIGA with different population sizes over (a) topology series #2 and (b) topology series #4.



(a)



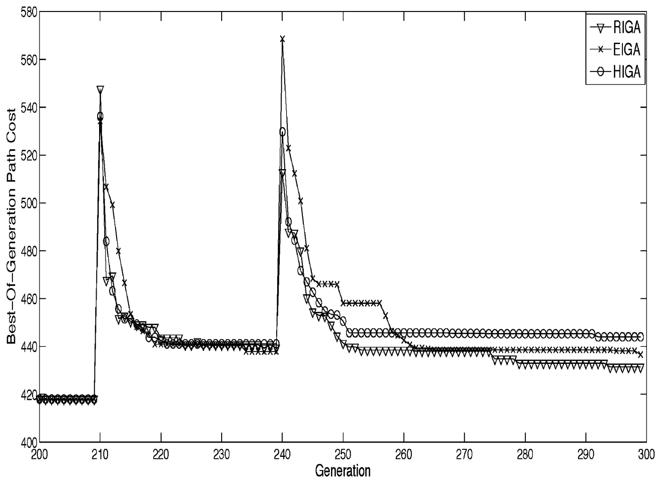
(b)

Fig. 4. Comparison results of the quality of solution for HIGA with different change intervals over (a) topology series #3 and (b) topology series #4.

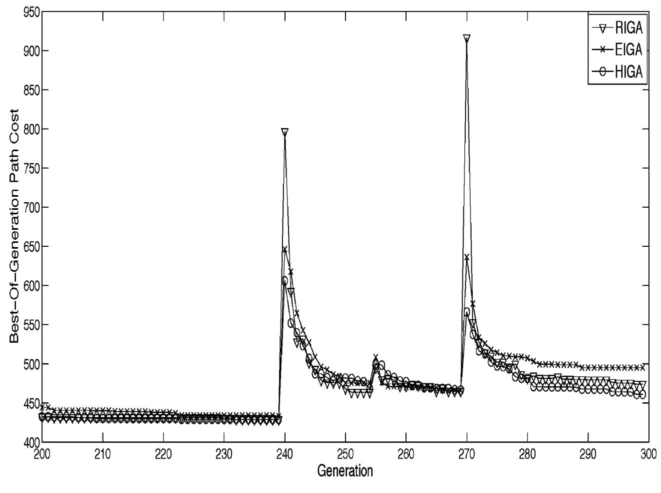
population evolves only five generations between two sequential changes. Intuitively, a larger interval will give the population more time to evolve and search better solutions than what a smaller interval does. We take HIGA as an example to compare the quality of solutions obtained at different intervals. However, one problem is that the total generations are different for different intervals, i.e., 105, 210, and 315 versus the interval 5, 10, and 15. Since the number of change points (i.e., the generation at which a new topology is applied) is the same for all the intervals, we take the data at each change point, and its left two and right two generations. Thus, the three datasets can be aligned over the three intervals. Fig. 4(a) and (b) shows the results over topology series #3 and #4, respectively. Since the generation number does not correspond to the actual number when the interval is 10 or 15, we rename it as pseudogeneration. From the two subfigures, it can be seen that the solution quality becomes better when the change interval becomes larger. Therefore, in a relatively slowly changing environment, the studied GAs can achieve a good performance.

Third, we investigate the impact of the change severity on the performance of algorithms. In our problem, the change severity is reflected by the number of nodes involved per change. Therefore, we choose topology series #2 and #4 as the two environments with different change severity. This time we pick up RIGA, EIGA, and HIGA together as the examples. To see the reaction of the algorithms to the changes clearly, we set the interval to 15. Fig. 5(a) and (b) shows the results over topology series #2 and #4, respectively.

From Fig. 5(a), it can be seen that there are two drastic change points: one is at generation 210 and the other is at generation 240. We count the number of generations that the population spends to find the best solution before the next change. For generation 210, it is 10, 9, and 11 for RIGA, EIGA, and HIGA, respectively. For generation 240, it is 13, 10, and 11 for them, respectively. The total average value is 10.67. From Fig. 5(b), we can see that there are three remarkable change points at generations 240, 255, and 270, respectively. We also count the aforementioned number. For generation 240, it is 11, 10, and

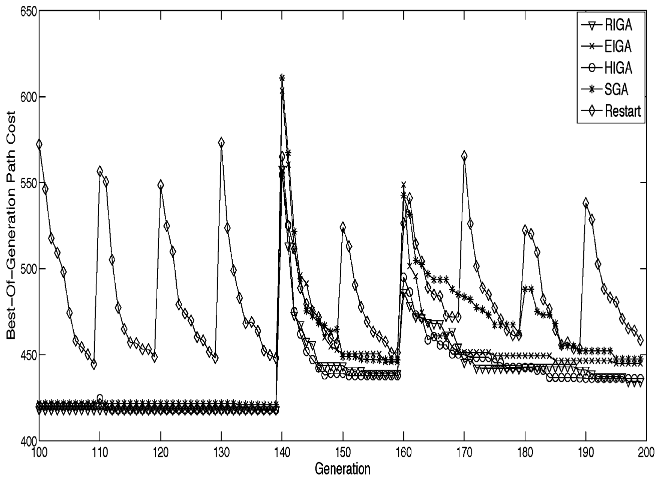


(a)

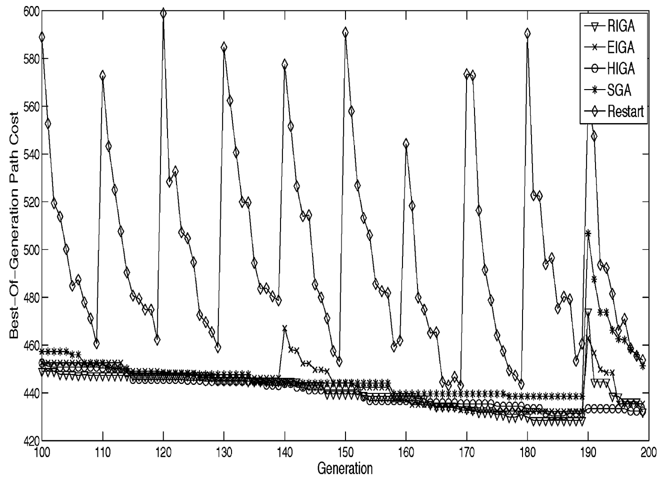


(b)

Fig. 5. Comparison results of the response speed to changes for RIGA, EIGA, and HIGA over (a) topology series #2 and (b) topology series #4.



(a)



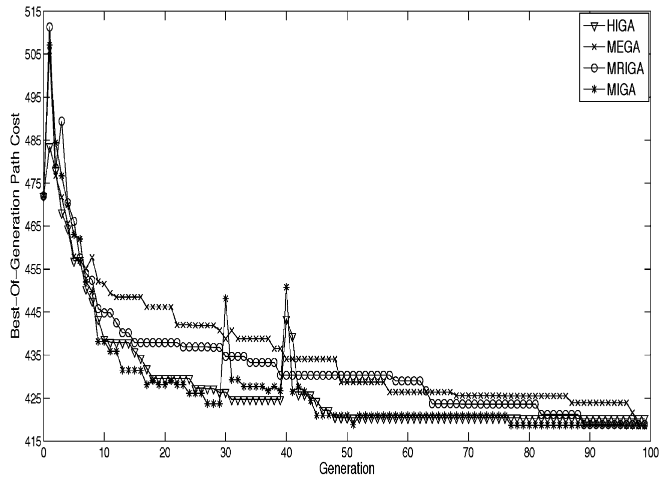
(b)

Fig. 6. Comparison results of the quality of solution for RIGA, EIGA, HIGA, SGA, and Restart GA over (a) topology series #2 and (b) topology series #3.

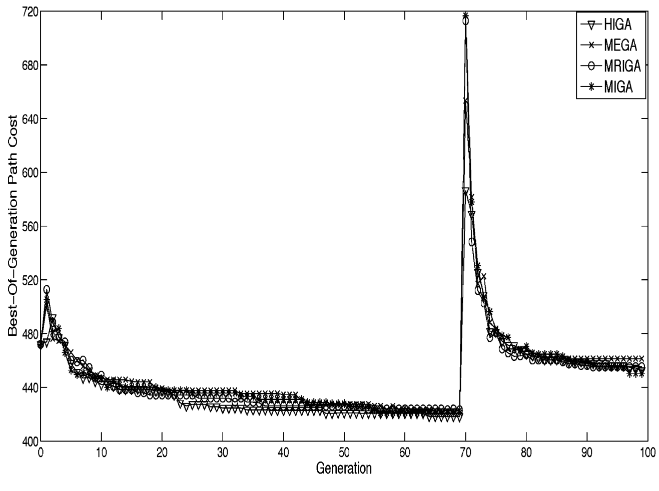
14 for RIGA, EIGA, and HIGA, respectively. For generation 255, it is 12, 13, and 14, respectively. For generation 270, it is 15, 12, and 13, respectively. The total average value is 12.67. On average, two more generations are spent for achieving good solutions in the topology series where more severe changes occur. Therefore, we can conclude that these dynamic GAs respond to the environmental changes at a reasonable speed, and the more severe the changes, the longer the response time.

Fourth, we compare the dynamic GAs with the traditional GAs over the dynamic SP problem. Since the dynamic GAs are designed for the dynamic environments, they should show better performance than the traditional GAs over our problem. We compare RIGA, EIGA, and HIGA with SGA and restart GA. We choose topology series #2 and #3 as the two dynamic environments. The interval is set to 10 here and also in the following experiments. Fig. 6(a) and (b) shows the comparison results over topology series #2 and #3, respectively. From Fig. 6(a) and (b), it can be seen that the restart GA exhibits the worst performance even when the changes have trivial impacts on the current population. The reason is that the restart GA does not exploit any useful information in the old environment and that the frequent restart sacrifices its evolving capability. Although SGA is much better than the restart GA, the best solutions that it can find in the new environment are not competitive to those found by any of the three GAs with immigrants schemes. The immigrants bring more diversity to the populations in RIGA, EIGA, and HIGA, and therefore, enhance their search capability.

Fifth, we compare the immigrants-based GAs with the memory-related GAs (i.e., MEGA, MRIGA, and MIGA) in the acyclic environments. According to the earlier experiments, HIGA is a good representative of the three immigrants-based GAs. Therefore, we evaluate the quality of solutions for HIGA, MEGA, MRIGA, and MIGA over topology series #2 and #3, respectively. The memory size is set to 20, and the reason will be described in the subsequent section. Fig. 7(a) and (b) shows the results. In Fig. 7(a), it can be seen that HIGA and MIGA show a competitive performance. MEGA performs the worst among all the memory-based GAs. The reason is that in our problem,



(a)



(b)

Fig. 7. Comparison results of the quality of solution for HIGA, MEGA, MRIGA, and MIGA over (a) topology series #2 and (b) topology series #3.

TABLE I
t-Test Results of Comparing GAs in Acyclic Dynamic Environments

t-test Result	Topology Series #2			Topology Series #3		
Environmental Dynamics R	5	10	15	5	10	15
$RIGA - SGA$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
$EIGA - SGA$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
$HIGA - SGA$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
$RIGA - HIGA$	$+$	$+$	$+$	$-$	$-$	$-$
$EIGA - HIGA$	$-$	$-$	$-$	$-$	$-$	$-$
$MEGA - HIGA$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
$MRIGA - HIGA$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
$MIGA - HIGA$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$

when a change occurs, the best individual in the memory may become infeasible. Therefore, the memory scheme may lose its power. However, in MRIGA and MIGA, the random immigrants are added or the individuals in the memory are just used to generate immigrants by mutation. Therefore, the infeasible solutions from the memory have a very high probability to be replaced by feasible solutions. In Fig. 7(b), it can be seen that although HIGA is not always the best, the memory degrades the algorithm performance when changes occur. Therefore, we conclude that the memory-related schemes have no advantages in acyclic dynamic environments.

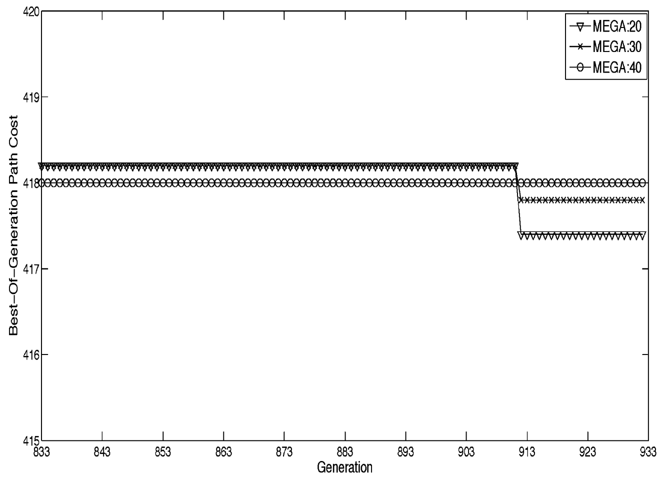
The corresponding statistical results of comparing these GAs in acyclic dynamic environments by a one-tailed t-test with 18 DOF at a 0.05 level of significance are given in Table I. In the table, the t-test result regarding Algorithm 1 and Algorithm 2 is shown as "+," "−," "$s+$," and "$s-$" when Algorithm 1 is insignificantly better than, insignificantly worse than, significantly better than, or significantly worse than Algorithm 2, respectively.

C. Experimental Results and Analysis in Cyclic Dynamic Environments

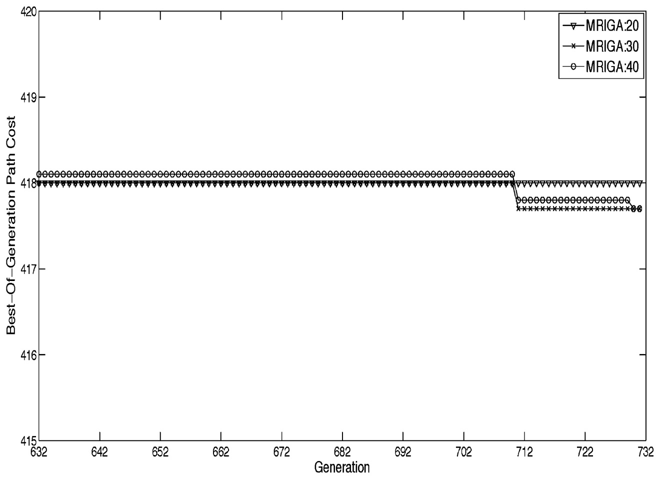
In this section, we focus on the cyclic dynamic environments, and topology series #1 builds such an environment for our experiments. First, we investigate the memory size that ensures a specified quality of solution for the memory-related schemes. We pick up MEGA and MRIGA as examples and run them on the cyclic topology series. Since there are 20 different topologies in this cyclic series, we set the minimum memory size to 20. Then we increase it to 30 and 40, respectively. We repeat 20 different toplogies five times, and the memory schemes will show more power when the same environments are visited more times. Therefore, we sample the data from the latter part of the evolutionary process in MEGA, MRIGA, and MIGA. Fig. 8(a) and (b) shows the results. From both subfigures, it can be seen that 20 is good enough in both MEGA and MRIGA. The increase of the memory size wastes more resources instead of benefiting the quality of solutions.

Second, we compare the memory-related schemes with the traditional GAs in the cyclic dynamic environment. Since the immigrants-based GAs beat both SGA and restart GA in the acyclic dynamic environment, we also want to know if the traditional GAs are suitable for cyclic dynamic environments. We evaluate MEGA, MRIGA, MIGA, SGA, and restart GA over topology series #1. Fig. 9(a) and (b) shows the results. From Fig. 9(a) and (b), it can be seen that the results are similar to the ones in Fig. 6. The restart GA always exhibits the worst performance. The frequent restart severely sacrifices its capability of searching the good solutions. Although SGA is much better than the restart GA, the best solutions that it can find in the new environment are not competitive to those found by any of the three memory-related GAs. Therefore, the traditional GAs do not work well in a cyclic dynamic environment either.

Third, we compare the three memory-related GAs with the immigrants-based GAs in the cyclic environments. We also pick up HIGA as the representative of the immigrants schemes. Fig. 10(a) and (b) shows the results. We can see that in both subfigures, the three memory-related schemes perform better than HIGA. It is contrary to the results shown in Fig. 7. From Fig. 10(a), it can be seen that when a change occurs, MEGA just takes one generation to find the good solution, while RIGA takes eight generations but still finds a worse solution. From Fig. 10(b), it can also be seen that MEGA, MRIGA, and MIGA also take one generation to find the good solution for the new

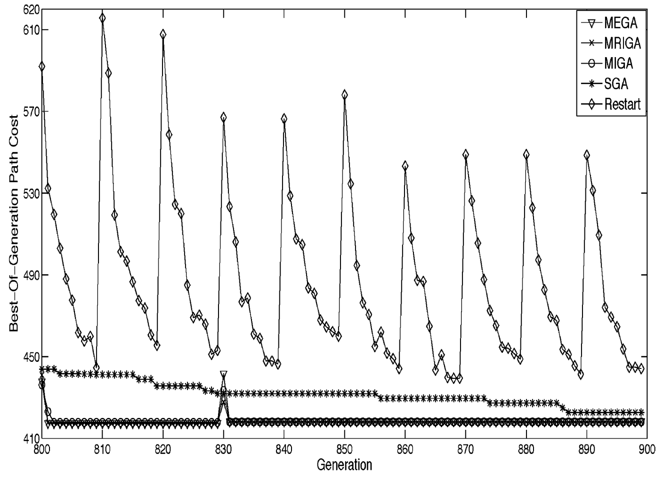


(a)

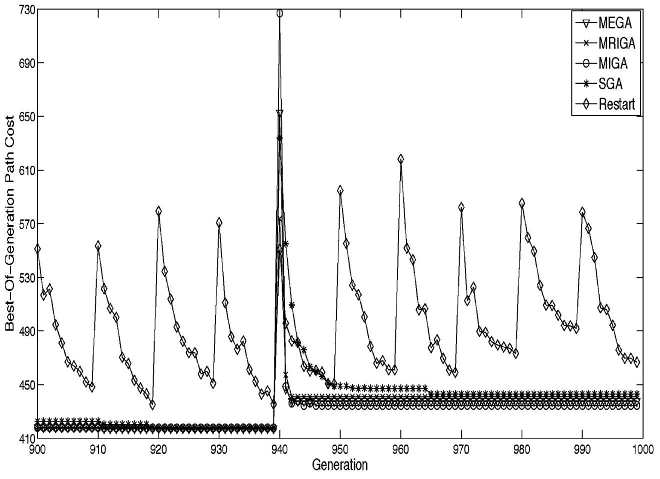


(b)

Fig. 8. Comparison results of the quality of solution under different memory sizes for (a) MEGA and (b) MRIGA.

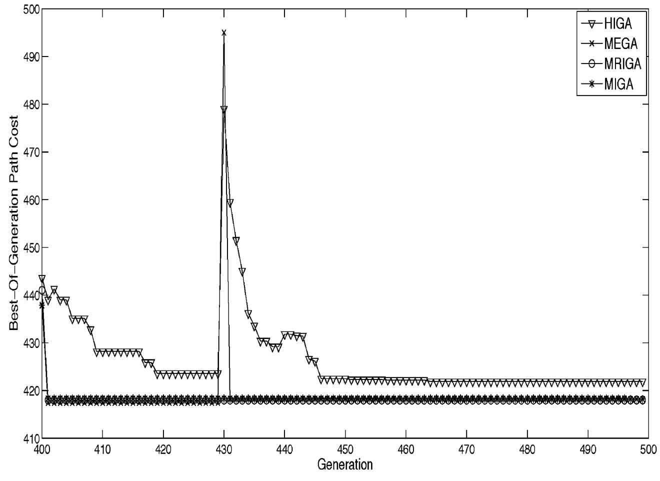


(a)

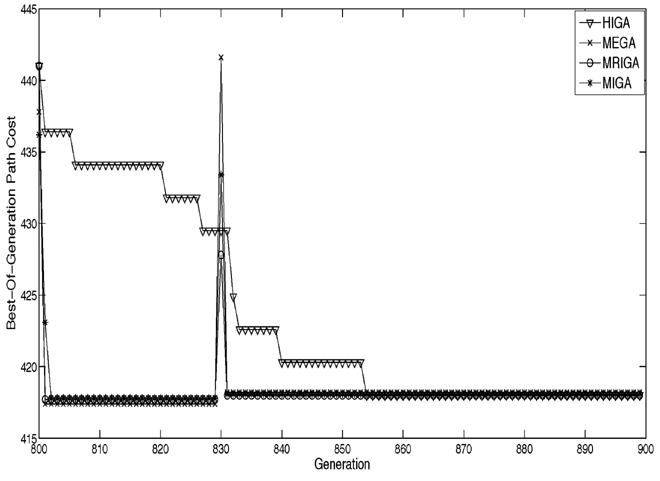


(b)

Fig. 9. Comparison results of the quality of solution for MEGA, MRIGA, MIGA, SGA, and restart GA in the cyclic topology series from (a) generation 800–899 and (b) generation 900–999.



(a)



(b)

Fig. 10. Comparison results of the quality of solution for HIGA, MEGA, MRIGA, and MIGA in the cyclic topology series from (a) generation 400– 499 and (b) generation 800–899.

TABLE II
t-Test Results of Comparing GAs in Cyclic Dynamic Environments

t-test Result	Topology Series #1		
Environmental Dynamics R	5	10	15
$MEGA - SGA$	$s+$	$s+$	$s+$
$MRIGA - SGA$	$s+$	$s+$	$s+$
$MIGA - SGA$	$s+$	$s+$	$s+$
$MEGA - HIGA$	$s+$	$s+$	$s+$
$MRIGA - HIGA$	$s+$	$s+$	$s+$
$MIGA - HIGA$	$s+$	$s+$	$s+$

environment. The reason is that the good solution stored in the memory can be retrieved immediately when the population enters an environment that has been visited before. Therefore, the memory-related schemes are really suitable for cyclic dynamic environments.

The corresponding statistical results of comparing these GAs in cyclic dynamic environment by a one-tailed t-test with 18 DOF at a 0.05 level of significance are given in Table II.

VII. Conclusion and Future Work

MANET is a self-organizing and self-configuring multihop wireless network, which has a wide usage nowadays. The SP routing problem aims to establish a multihop forwarding path from a source node to a destination node and is one important issue that significantly affects the performance of MANETs. So far, most SP routing algorithms in the literature consider only the fixed network topology. It is much more challenging to deal with the SP routing problem in a continuously changing network like MANETs than to solve the static one in a fixed infrastructure. In recent years, there has been a growing interest in studying GAs for DOPs. Among approaches developed for GAs to deal with DOPs, immigrants schemes aim at maintaining the diversity of the population throughout the run via introducing random individuals into the current population, while memory schemes aim at storing useful information for possible reuse in a cyclic dynamic environment.

This paper investigates the application of GAs for solving the DSPRP in MANETs. A DSPRP model is built up in this paper. A specialized GA is designed for the SP problem in MANETs. Several immigrants and/or memory schemes that have been developed for GAs for general DOPs are adapted and integrated into the specialized GA (which gives several GA variants) to solve the DSPRP in MANETs. Then, extensive simulation experiments are conducted based on a large-scale MANET constructed in this paper to evaluate various aspects of these GA variants for the DSPRP. The experimental results indicate that both immigrants and memory schemes enhance the performance of GAs for the DSPRP in MANETs.

Generally speaking, the immigrants schemes show their power in acyclic dynamic environments, and the memory-related schemes beat other schemes in cyclic dynamic environments.

We believe that this is the first study that investigates the effectiveness and efficiency of GAs with immigrants and memory schemes in solving the DSPRP in the real-world networks, i.e., MANETs. There are several relevant future works. One interesting work is to further investigate other approaches developed for GAs for general DOPs to solve the DSPRP in MANETs and other relevant networks [33]. Another future work is to investigate the application of GAs studied in this paper for solving other dynamic routing problems in MANETs. For example, applying GA approaches for the multicasting routing problem [7] in dynamic network environments is to be investigated in the future.

Acknowledgment

The authors are grateful to the anonymous associate editor and reviewers for their thoughtful suggestions and constructive comments.

References

[1] C. W. Ahn, R. S. Ramakrishna, C. G. Kang, and I. C. Choi, “Shortest path routing algorithm using Hopfield neural network,” *Electron. Lett.*, vol. 37, no. 19, pp. 1176–1178, Sep. 2001.

[2] C. W. Ahn and R. S. Ramakrishna, “A genetic algorithm for shortest path routing problem and the sizing of populations,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 6, pp. 566–579, Dec. 2002.

[3] M. K. Ali and F. Kamoun, “Neural networks for shortest path computation and routing in computer networks,” *IEEE Trans. Neural Netw.*, vol. 4, no. 6, pp. 941–954, Nov. 1993.

[4] J. Branke, “Memory enhanced evolutionary algorithms for changing optimization problems,” in *Proc. 1999 Congr. Evol. Comput.*, pp. 1875–1882.

[5] J. Branke, *Evolutionary Optimization in Dynamic Environments*. Norwell, MA: Kluwer, 2002.

[6] J. Branke, T. Kaußler, C. Schmidt, and H. Schmeck, “A multi-population approach to dynamic optimization problems,” in *Proc. 4th Int. Conf. Adaptive Comput. Des. Manuf.*, 2000, pp. 299–308.

[7] H. Cheng, X. Wang, S. Yang, and M. Huang, “A multipopulation parallel genetic simulated annealing based QoS routing and wavelength assignment integration algorithm for multicast in optical networks,” *Appl. Soft Comput.*, vol. 9, no. 2, pp. 677–684, Mar. 2009.

[8] H. G. Cobb and J. J. Grefenstette, “Genetic algorithms for tracking changing environments,” in *Proc. 5th Int. Conf. Genet. Algorithms*, 1993, pp. 523–530.

[9] D. Dasgupta and D. McGregor, “Nonstationary function optimization using the structured genetic algorithm,” in *Proc. 2nd Int. Conf. Parallel Problem Solving Nature*, 1992, pp. 145–154.

[10] A. Das and C. Martel, “Stochastic shortest path with unlimited hops,” *Inf. Process. Lett.*, vol. 109, no. 5, pp. 290–295, 2009.

[11] D. Din, “Anycast routing and wavelength assignment problem on WDM network,” *IEICE Trans. Commun.*, vol. E88-B, no. 10, pp. 3941–3951, Oct. 2005.

[12] J. J. Grefenstette, “Genetic algorithms for changing environments,” in *Proc. 2nd Int. Conf. Parallel Problem Solving Nature*, 1992, pp. 137–144.

[13] S. Lee, S. Soak, K. Kim, H. Park, and M. Jeon, “Statistical properties analysis of real world tournament selection in genetic algorithms,” *Appl. Intell.*, vol. 28, no. 2, pp. 195–2205, Apr. 2008.

[14] E. Lewis and G. Ritchie, “A comparison of dominance mechanisms and simple mutation on non-stationary problems,” in *Proc. 5th Int. Conf. Parallel Problem Solving Nature*, 1998, pp. 139–148.

[15] S. Louis and Z. Xu, “Genetic algorithms for open shop scheduling and re-scheduling,” in *Proc. 11th ISCA Int. Conf. Comput. Appl.*, 1996, pp. 99–102.

[16] S. Misra and B. J. Oommen, “GPSPA: A new adaptive algorithm for maintaining shortest path routing trees in stochastic networks,” *Int. J. Commun. Syst.*, vol. 17, no. 10, pp. 963–984, 2004.

[17] A. W. Mohemmed, N. C. Sahoo, and T. K. Geok, “Solving shortest path problem using particle swarm optimization,” *Appl. Soft Comput.*, vol. 8, no. 4, pp. 1643–1653, Sep. 2008.

[18] H. Mori and Y. Nishikawa, “Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm,” in *Proc. 7th Int. Conf. Genet. Algorithms*, 1997, pp. 299–306.

[19] R. W. Morrison, *Designing Evolutionary Algorithms for Dynamic Environments*. Berlin, Germany: Springer-Verlag, 2004.

[20] R. W. Morrison and K. A. De Jong, "Triggered hypermutation revisited," *Proc. 2000 Congr. Evol. Comput.*, vol. 2, pp. 1025–1032.

[21] S. Oh, C. Ahn, and R. Ramakrishna, "A genetic-inspired multicast routing optimization algorithm with bandwidth and end-to-end delay constraints," in *Proc. 13th Int. Conf. Neural Inf. Process.* (Lecture Notes in Computer Science), 2006, vol. 4234, pp. 807–816.

[22] F. Oppacher and M. Wineberg, "The shifting balance genetic algorithm: Improving the GA in a dynamic environment," *Proc. 1999 Genet. Evol. Comput. Conf.*, vol. 1, pp. 504–510.

[23] D. Parrott and X. Li, "Locating and tracking multiple dynamic optima by a particle swarm model using speciation," *IEEE Trans. Evol. Comput.*, vol. 10, no. 4, pp. 440–458, Aug. 2006.

[24] M. Parsa, Q. Zhu, and J. Garcia-Luna Aceves, "An iterative algorithm for delay-constrained minimum-cost multicasting," *IEEE/ACM Trans. Netw.*, vol. 6, no. 4, pp. 461–474, Aug. 1998.

[25] C. E. Perkins Ed., *Ad Hoc Networking*. London, U.K.: Addison-Wesley, 2001.

[26] C. S. R. Murthy and B. S. Manoj, *Ad Hoc Wireless Networks: Architectures and Protocols*. Englewood Cliffs, NJ: Prentice-Hall, 2004.

[27] R. Tinos and S. Yang, "A self-organizing random immigrants genetic algorithm for dynamic optimization problems," *Genet. Program. Evol. Mach.*, vol. 8, no. 3, pp. 255–286, Sep. 2007.

[28] C.-K. Toh, *Ad Hoc Mobile Wireless Networks: Protocols and Systems*. Englewood Cliffs, NJ: Prentice-Hall, 2002.

[29] K. Trojanowski and Z. Michalewicz, "Searching for optima in non-stationary environments," in *Proc. 1999 Congr. Evol. Comput.*, vol. 3, pp. 1843–1850.

[30] K. Trojanowski and Z. Michalewicz, "Evolutionary optimization in non-stationary environments," *J. Comput. Sci. Technol.*, vol. 1, no. 2, pp. 93–124, 2000.

[31] A. Uyar and A. Harmanci, "A new population based adaptive dominance change mechanism for diploid genetic algorithms in dynamic environments," *Soft Comput.*, vol. 9, no. 11, pp. 803–815, Nov. 2005.

[32] F. Vavak and T. C. Fogarty, "A comparative study of steady state and generational genetic algorithms for use in nonstationary environments," in *Proc. AISB Workshop Evol. Comput.*, 1996, pp. 297–304.

[33] Y. Xu, S. Salcedo Sanz, and X. Yao, "Metaheuristic approaches to traffic grooming in WDM optical networks," *Int. J. Comput. Intell. Appl.*, vol. 5, no. 2, pp. 231–249, Jun. 2005.

[34] S. Yang, "Memory-based immigrants for genetic algorithms in dynamic environments," in *Proc. 2005 Genet. Evol. Comput. Conf.*, vol. 2, pp. 1115–1122.

[35] S. Yang, "Population-based incremental learning with memory scheme for changing environments," in *Proc. 2005 Genet. Evol. Comput. Conf.*, vol. 1, pp. 711–718.

[36] S. Yang, "Genetic algorithms with elitism-based immigrants for changing optimization problems," in *Proc. EvoWorkshops 2007: Appl. Evol. Comput.* (Lecture Notes in Computer Science), vol. 4448, pp. 627–636.

[37] S. Yang, "Genetic algorithms with memory- and elitism-based immigrants in dynamic environments," *Evol. Comput.*, vol. 16, no. 3, pp. 385–416, Sep. 2008.

[38] S. Yang, Y.-S. Ong, and Y. Jin, Eds., *Evolutionary Computation in Dynamic and Uncertain Environments*. Berlin, Germany: Springer-Verlag, 2007.

[39] S. Yang and H. Richter, "Hyper-learning for population-based incremental learning in dynamic environments," in *Proc. 2009 Congr. Evol. Comput.*, pp. 682–689.

[40] S. Yang and R. Tinos, "A hybrid immigrants scheme for genetic algorithms in dynamic environments," *Int. J. Autom. Comput.*, vol. 4, no. 3, pp. 243–254, Jul. 2007.

[41] S. Yang and R. Tinos, "Hyper-selection in dynamic environments," in *Proc. 2008 Congr. Evol. Comput.*, pp. 3185–3192.

[42] S. Yang and X. Yao, "Experimental study on population-based incremental learning algorithms for dynamic optimization problems," *Soft Comput.*, vol. 9, no. 11, pp. 815–834, Nov. 2005.

[43] S. Yang and X. Yao, "Population-based incremental learning with associative memory for dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 12, no. 5, pp. 542–561, Oct. 2008.

[44] X. Yu, K. Tang, T. Chen, and X. Yao, "Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization," *Memetic Comput.*, vol. 1, no. 1, pp. 3–24, Mar. 2009.

[45] X. Yu, K. Tang, and X. Yao, "An immigrants scheme based on environmental information for genetic algorithms in changing environments," in *Proc. 2008 Congr. Evol. Comput.*, pp. 1141–1147.

Shengxiang Yang (M'00) received the B.Sc. and M.Sc. degrees in automatic control and the Ph.D. degree in systems engineering from Northeastern University, Shenyang, China, in 1993, 1996, and 1999, respectively.

From October 1999 to October 2000, he was a Postdoctoral Research Associate with the Algorithm Design Group, Department of Computer Science, King's College London, London, U.K. He is currently a Lecturer with the Department of Computer Science, University of Leicester, Leicester, U.K. He is the author or coauthor of more than 90 publications. He is the area editor, an associate editor, or a member of the editorial boards of four international journals. He has coedited several books and conference proceedings, and co-guest-edited special issuaes of several journals. His current research interests include evolutionary algorithms, swarm intelligence, meta-heuristics and hyper-heuristics, artificial neural networks, computational intelligence in dynamic and uncertain environments, scheduling, network flow problems and algorithms, and real-world applications.

Dr. Yang is a Member of the Association of Computing Machinery Special Interest Group on Genetic and Evolutionary Computation. He is also a Member of the Task Force on Evolutionary Computation in Dynamic and Uncertain Environments, Evolutionary Computation Technical Committee, IEEE Computational Intelligence Society. He has given invited keynote speeches at several international conferences, and has co-organized several workshops and special sessions in conferences.

Hui Cheng received the B.Sc. and M.Sc. degrees in computer science from Northeastern University, Shenyang, China in 2001 and 2004, respectively, and the Ph.D. degree in computer science from Hong Kong Polytechnic University, Kowloon, Hong Kong, in 2007.

He is currently a Postdoctoral Research Associate with the Department of Computer Science, University of Leicester, Leicester, U.K. His current research interests include evolutionary computation for dynamic optimization problems, mobile *ad hoc* networks, multicast routing, and quality-of-service mobile group communication. He is the author or coauthor of more than 30 technical papers.

Fang Wang (M'01) received the B.Sc. and M.Sc. degrees in computer science from China, and the Ph.D. degree in artificial intelligence from the University of Edinburgh, Edinburgh, U.K.

She was an Assistant Professor in China. In 2000, she joined British Telecommunications Innovate, Ipswich, U.K., where she is currently a Senior Research Scientist with the Centre for Information and Security Systems Research. Her current research interests include software agents, cognitive neuroscience, and distributed computing. She is the author or coauthor of several book chapters, and journal and conference papers. She holds a number of patents.

Dr. Wang has received several technical awards.