2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

# An Improved Genetic Algorithm for Dynamic Shortest Path Problems

Xuezhi Zhu[1, 2], Wenjian Luo[1, 2, *] and Tao Zhu[1, 2]

[1]School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, Anhui, China
[2]Anhui Province key Laboratory of Software Engineering in Computing and Communication, University of Science and Technology of China, Hefei 230027, Anhui, China
zxz1990@mail.ustc.edu.cn, wjluo@ustc.edu.cn, zhuta@mail.ustc.edu.cn

***Abstract*—The Shortest Path (SP) problems are conventional combinatorial optimization problems. There are many deterministic algorithms for solving the shortest path problems in static topologies. However, in dynamic topologies, these deterministic algorithms are not efficient due to the necessity of restart. In this paper, an improved Genetic Algorithm (GA) with four local search operators for Dynamic Shortest Path (DSP) problems is proposed. The local search operators are inspired by Dijkstra's Algorithm and carried out when the topology changes to generate local shortest path trees, which are used to promote the performance of the individuals in the population. The experimental results show that the proposed algorithm could obtain the solutions which adapt to new environments rapidly and produce high-quality solutions after environmental changes.**

## I. INTRODUCTION

The Shortest Path (SP) problems are traditional combinatorial optimization problems. There are several kinds of SP problems such as the Single-Source Shortest Path (SSSP) problems, the Multi-Source Shortest Path (MSSP) problems the Static Shortest Path (SSP) problems and the Dynamic Shortest Path (DSP) problems. Given a network topology with some nodes and some weighted edges, the SP problem aims to find a path between two nodes so that the sum of the cost of its edges is minimal. There are many deterministic algorithms for solving the SP problems, such as Dijkstra's Algorithm [1] and Bellman-Ford Algorithm [2]. They work well in static topologies, namely the topologies are fixed. However, in dynamic topologies, these deterministic algorithms are not efficient due to the necessity of restart [3]. Therefore, for DSP problems, new solving methods are needed. Fortunately, Genetic Algorithms (GAs) perform well for solving Dynamic Optimization Problems (DOPs).

A GA, which is based on the mechanisms of biological evolution, is a kind of stochastic algorithms for global optimization [4]. A GA mainly includes chromosome coding, initialization, evaluation, selection, crossover and mutation [5]. By the principle of "survival of the fittest", the individuals could get close to optima by some generations of evolution [6]. Based on the basic GA, some new GAs for solving DOPs have been proposed. And typical GAs for DOPs are described as follows. GAs with elitism-based immigrants are proposed in [7]. The elitism-based immigrants are based on the elite in the previous generation, and therefore, the population is guided by these immigrants. Besides, a self-organizing random immigrants GA is proposed in [8], where the immigrants are randomly generated. And a hybrid immigrants scheme for GAs is proposed in [9]. The memory-based immigrants for GAs are adopted in [10] and [11], where the suitable immigrants stored in the memory could be used for the current population when the environment changes. Both the memory-based and the elitism-based immigrants are adopted for the GAs in [12].

For DSP problems, an improved genetic algorithm for DSP problems is proposed in this paper, which combines a GA in [3] with four local search operators inspired by Dijkstra's Algorithm. Dijkstra's Algorithm is a simple and efficient deterministic algorithm for SP problems, but it is not suitable for dynamic environments because it has to rerun when the environment changes. GAs perform well in dynamic environments, but the solutions they get are not optimal sometimes. Therefore, to enhance the search ability of GAs, four local search operators inspired by Dijkstra's Algorithm are designed to search around some nodes (the source node, the destination node and the nodes affected by the environmental changes) when the initial population is generated or the topology changes. By the local search operators, local shortest path trees are generated and used to promote the performance of the individuals in the population in the evolution stages. So that the proposed algorithm could get solutions which adapt to new environments rapidly and produce high-quality solutions after environmental changes.

The rest of this paper is organized as follows. The description of the DSP problem and the related work are introduced in section II. In section III, the proposed algorithm is presented. The experimental results and the analyses are given in section IV. Section V concludes this paper and introduces the future work.

## II. RELATED WORK

### A. The DSP Problem

The DSP model we consider in this paper is described as follows. Given a real-time network topology $G(V, E, t)$ where $V$ stands for the nodes and $E$ stands for the weighted edges at time $t$. The DSP problem aims to find a path from a source node $s$ to a destination node $d$ so that the sum of the cost of

*Corresponding author.
Tel. : +86 – 551 - 63602824
978-1-4799-1488-3/14/$31.00 ©2014 IEEE

its edges is minimized at time $t$. Here, the simulated dynamic environment is the same as that in [3], where some nodes are sleeping and wakening over time. The formalized description of the DSP problem is given as follows.

Here are some symbols we use for the problem:

| | |
|---|---|
| $G(V, E, t)$ | the topology graph at time $t$; |
| $N$ | the total number of the nodes in $G$; |
| $V$ | the nodes in $G$, namely $\{v_1, v_2, \ldots, v_N\}$; |
| $E$ | the weighted edges in $G$, such as $(v_i, v_j)$; |
| $s$ | the source node; |
| $d$ | the destination node; |
| $cost(v_i, v_j)$ | the cost on the edge $(v_i, v_j)$; |
| $P(s, d, t)$ | the path from $s$ to $d$ at time $t$; |
| $C(P)$ | the total cost of the path $P$. |

The purpose of the DSP problem is to find a path $P(s, d, t)$ whose cost $C(P)$ is minimized at time $t$, as is shown in (1).

$$C(P) = \min_{P \in G} \left\{ \sum_{(v_i, v_j) \in P(s,d,t)} cost(v_i, v_j) \right\} \quad (1)$$

### B. The GAs for the SSP Problems

There are some GAs solving the SSP problems. A GA for the SP problem is proposed in [13], in which a path is represented by a variable-length chromosome and a repair method is proposed to eliminate the loops in a path. And the simulated experiments show that the GA has lower failure ratio and higher rate of convergence than other GAs. The feasibility of using GAs to solve SP problems is investigated in [14], and a priority-based approach is proposed to encode all the potential paths. The priority-based encoding method is also adopted in [15], where the priority is represented by random keys and arithmetical crossover, swap mutation and immigration are adopted.

### C. The GAs for the DSP Problems

The DSP problems are more complex than the SSP problems due to the dynamics of environments. Therefore, the algorithms for the SSP problems are not suitable for dynamic environments generally. An elitism-based GA for the DSP problems is implemented in [16]. Firstly, the specialized GA for SP is designed, and then in each generation, a certain number of elitism-based immigrants are produced and added to the population to maintain the diversity. The immigrants could provide the guidance to search good solutions. The GAs with immigrants and memory schemes are investigated in [3]. Once the topology changes, the new immigrants and the information in the memory will help the population adapt to the new environment. Multi-population GAs with immigrants schemes are proposed in [17]. The search space is divided into several parts and several small populations are needed for the partial spaces. In [17], after a certain number of generations, the parent population is divided into three small populations, one of them is designed to explore, and the others aim to exploit. Then all the small populations will be merged when the topology changes.

### D. The Specialized GA for the SP Problem

Our algorithm is based on the specialized GA for the SP problem, which is proposed in [13] and adopted in [3]. Therefore, the procedures of the GA, including chromosome representation, population initialization, fitness function, selection, crossover, mutation and the repair method are described as follows.

#### 1) Chromosome representation

A chromosome represents a solution, which is a path from $s$ to $d$ in the DSP problem. The chromosome for an individual in this paper is represented by a string with variable length [13]. Each locus in the string stores a positive integer which represents a node ID. The first locus and the last locus represent $s$ and $d$, respectively. The other loci represent the intermediate nodes in the path from $s$ to $d$. For example, the chromosome $(s, v_i, v_j, \ldots, d)$ represents a path $P(s, d, t)$ through $v_i$ and $v_j$. As a result, the length of a chromosome is not greater than $N$, and the continuous two nodes in a chromosome are connected in the topology.

#### 2) Population initialization

The initialization of the population is the first step in GA. There are mainly two kinds of initialization, namely random initialization and heuristic initialization [13]. The random initialization is adopted in this paper to increase the diversity of the initial population. The details of the process to initialize a chromosome are described as follows. Firstly, $s$ is stored in the first locus of the chromosome. Then, one of the adjacent nodes of $s$ is randomly selected (say $v_i$) and stored in the second locus of the chromosome. And then, one of the adjacent nodes of $v_i$ is randomly selected. These processes are repeated until $d$ is selected. Notice that: (i) Each node will be selected once at most to avoid loops; (ii) If there is no alternative adjacent node to be selected for a chromosome, the chromosome will be reinitialized.

#### 3) Fitness function

The fitness function is used to evaluate the performance of the individuals. For the DSP problem, a path with the less total cost has the higher fitness. Hence, the fitness for the individual *ind,* which represents the path $P(s, d, t)$, is shown in (2).

$$fitness(ind) = \left[ \sum_{(v_i, v_j) \in P(s,d,t)} cost(v_i, v_j) \right]^{-1} \quad (2)$$

#### 4) Selection

By the selection, the individuals with high fitness are reserved while those with low fitness may be eliminated. For the sake of simplicity and high-efficiency, the pairwise tournament selection without replacement is adopted [18], namely two individuals are picked out and the one with the higher fitness will be selected, while the same individual should not be picked out twice.

#### 5) Crossover

Crossover, which is one of the core operators in the GA, is a procedure for genetic recombination. The crossover in this paper is a variant of one-point crossover [19]. The details for the crossover are described as follows. Two individuals for crossover are randomly picked, say $ind_1$ and $ind_2$, which represent the paths $P_1(s, d, t)$ and $P_2(s, d, t)$, respectively.

Notice that $P_1$ and $P_2$ must have at least one common intermediate node (denoted by $v_{mid}$, if there are two or more common intermediate nodes, one of them is randomly selected). Then $ind_1$ can be viewed as two sub-paths $P_1(s, v_{mid}, t) + P_1(v_{mid}, d, t)$. Similarly, $ind_2$ can be viewed as two sub-paths $P_2(s, v_{mid}, t) + P_2(v_{mid}, d, t)$. Through the crossover, two individuals (denoted by $ind_3$ and $ind_4$) are generated. The individual $ind_3$ represents the path $P_1(s, v_{mid}, t) + P_2(v_{mid}, d, t)$, and $ind_4$ represents the path $P_2(s, v_{mid}, t) + P_1(v_{mid}, d, t)$.

*6) Mutation*

The diversity of the population could be increased by mutation [4]. The mutation operator in this paper is described below. For an individual (denoted by *ind*) which represents the path $P(s, d, t)$ to be mutated, a locus (say $v_r$) in the chromosome is randomly selected. Then *ind* can be viewed as two sub-paths $P(s, v_r, t) + P(v_r, d, t)$. The mutation operator for *ind* produces an individual $ind_m$, which represents the path $P(s, v_r, t) + P_{rand}(v_r, d, t)$. Here, $P_{rand}(v_r, d, t)$ is a path from $v_r$ to $d$ which is randomly generated using the same method in the population initialization procedure.

*7) Repair method*

The crossover operator may produce the paths with loops, while for the initialization and the mutation above, there are no paths with loops due to the avoidance of generating loops. In this paper, the repair method proposed in [13] is adopted to eliminate loops. Assume that the individual *ind*, which represents the path from $s$ to $d$: $P(s, v_c, t) + P_c(v_c, v_c, t) + P(v_c, d, t)$. The sub-path $P_c(v_c, v_c, t)$ is a loop. The repair method is designed to delete the sub-path $P_c(v_c, v_c, t)$. That is, the repaired individual represents the path $P(s, v_c, t) + P(v_c, d, t)$.

## III. The Proposed Algorithm

In this section, we mainly introduce the improvement based on the GA described in section II-D. Because GAs adapt to dynamic environments rapidly and Dijkstra's Algorithm can find the deterministic SP efficiently, an improved GA is proposed by combining the advantages of both GAs and Dijkstra's Algorithm. The proposed algorithm adopts four local search operators inspired by Dijkstra's Algorithm to perform local searches for some nodes when the topology changes. These nodes are the source node, the destination node and the nodes affected by the environmental changes. By these local searches, some shortest path trees are obtained. Then the obtained shortest path trees are used to enhance the performance of the population in the evolution process. The details for the proposed GA are described as follows.

### *A. Local Search Operators for s and d*

It is obviously known that the shortest path from $s$ to $d$ contains $s$ and $d$. Hence, the local searches for $s$ and $d$ could enhance the individuals well. The local search means a procedure inspired by Dijkstra's Algorithm, namely the procedure will be terminated after searching for a certain number of nodes by the basic steps of Dijkstra's Algorithm.

When the algorithm starts or the environment changes, s is regarded as the source node and a certain number (denoted by $num_s$) of nodes are searched using the local search procedure. Then a local shortest path tree (denoted by $SPTree_s$), of which $s$ is the root, is obtained. And the path from $s$ to a node (denoted by $v_i$) in the $SPTree_s$ is the shortest path from $s$ to $v_i$. Similarly, $d$ is regarded as the source node and $num_d$ nodes are searched using the local search procedure. A local shortest path tree (denoted by $SPTree_d$) is obtained.

In the evolution process, for each individual, if a non-root node (denoted by $v_1$, if there are two or more nodes, one is randomly selected, the same below) in the $SPTree_s$ appears in the individual (denoted by *ind*), the sub-path from $s$ to $v_1$ in *ind* is replaced by the corresponding sub-path in the $SPTree_s$. Similarly, if a non-root node (denoted by $v_2$) in the $SPTree_d$ appears in *ind*, the sub-path from $v_2$ to $d$ in *ind* is replaced by the corresponding sub-path in the $SPTree_d$. And a new individual (say $ind_{new}$) is generated after the above processes, and an example is shown in Fig. 1. If $ind_{new}$ contains loops, the repair method in section II-D is performed. And the generated individual $ind_{new}$ is added into the set of improved individuals (denoted by *ImpIndSet*) to be used.

### *B. Local Search Operator for New Wake-up Nodes*

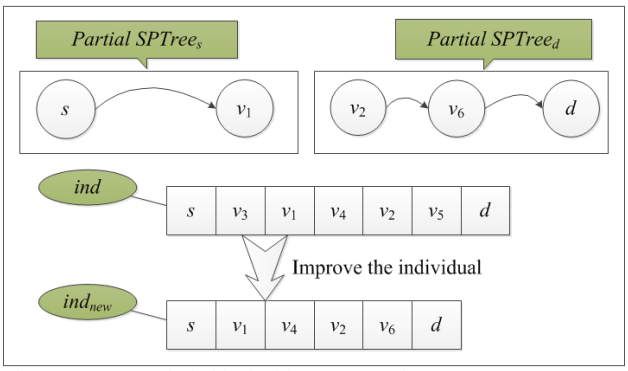



Fig. 1. Improve an individual with $SPTree_s$ and $SPTree_d$

For the dynamic environments we consider, if some sleeping nodes wake up, the topology changes. If a node (denoted by $v_w$) wakes up, $v_w$ is regarded as the source node for the local search and a certain number (denoted by $num_{waking}$) of nodes are searched. And a local shortest path tree (denoted by $SPTree_{waking}$), of which $v_w$ is the root, is produced.

In the evolution process, for each individual, if two nodes (denoted by $v_1$ and $v_2$, if there are more than two nodes, two among them are randomly selected) in the $SPTree_{waking}$ appear in the individual (denoted by *ind*), the sub-path from

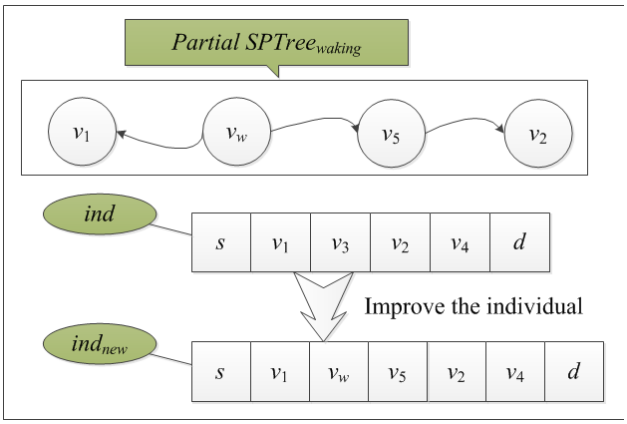



Fig. 2. Improve an individual with $SPTree_{waking}$

$v_1$ to $v_2$ in *ind* is replaced by the corresponding sub-path in the $SPTree_{waking}$, and a new individual (say $ind_{new}$) is generated, and an example is shown in Fig. 2. As is mentioned above, if $ind_{new}$ contains loops, the repair method in section II-D is performed. And $ind_{new}$ is added into *ImpIndSet*.

### C. Local Search Operator for Sleeping Nodes

If some active nodes sleep, the topology changes. Some individuals in the population may become infeasible solutions. In this case, a reconstruction method is proposed in this paper. For each individual, if a node (denoted by $v_1$) becomes sleeping, a node (denoted by $v_2$) in the path from $s$ to $v_1$ in the individual (denoted by *ind*) is randomly selected. Then $v_2$ is regarded as the source node for the local search, and a certain number (say $num_{sleeping}$) of nodes are searched using the local search procedure in section III-A. And a local shortest path (denoted by $SPTree_{sleeping}$) whose root is $v_2$, are produced. Then, for each node in the sub-path from $v_1$ to $d$ in *ind*, if there exists a node (say $v_3$) in the $SPTree_{sleeping}$, the sub-path from $v_2$ to $v_3$ in *ind* is replaced by that in the $SPTree_{sleeping}$, otherwise the sub-path from $v_2$ to $v_3$ in *ind* is replaced by a randomly generated sub-path. And a new individual (denoted by $ind_{new}$) is generated, and an example is shown in Fig. 3. If $ind_{new}$ contains loops, it will be repaired as is mentioned in section II-D. Then *ind* is replaced by the new generated individual $ind_{new}$ directly since *ind* is infeasible for the current environment.

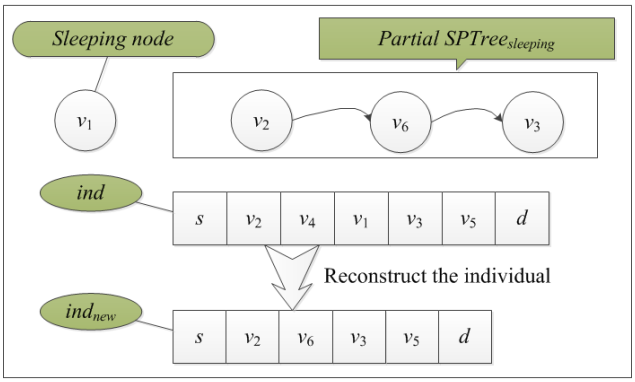


Fig. 3. Reconstruct an individual containing a sleeping node

### D. The Outline of the Proposed Algorithm

In order to conveniently control the parameters, the above numbers $num_s$, $num_d$, $num_{waking}$ and $num_{sleeping}$ for local searches are set to the same value. In addition to the procedures mentioned above, the scheme of elitism-based immigrants in [3] is adopted in this paper to enhance the performance. The algorithm framework is shown in Algorithm 1.

In Algorithm 1, the shortest path trees $SPTree_s$, $SPTree_d$, $SPTree_{waking}$ and $SPTree_{sleeping}$ are generated using the local search operators mentioned above. The steps from 5 to 21 mean the processes for environmental changes. Step 5 is called for $s$ and $d$. Step 7 is called for the new wake-up nodes. Steps 10-20, which are the reconstruction method described in section III-C, are called for the new sleeping nodes. The sign *popsize* means the population size and $Path(v_j, v_k, container)$ means the sub-path from $v_j$ to $v_k$ in *container*, where *container* stands for $SPTree_s$, $SPTree_d$, $SPTree_{waking}$, $SPTree_{sleeping}$ or an individual. The set of improved individuals *ImpIndSet* is cleared in step 23. The $SPTree_s$ and $SPTree_d$ are used to improve the individuals in the population (steps 24-33), and the improved individuals are added into *ImpIndSet*. And the similar processes are carried out using $SPTree_{waking}$ (steps 35-40). The basic operators for a GA are called (steps 42-46), and the elitism-based immigrants operation is performed (step 43). Steps 47 and 48 aim to remove the redundant individuals and add the improved individuals from *ImpIndSet* into the population. The details for steps 47 and 48 are described as follows.

**Algorithm 1.** The Improved Genetic Algorithm for the DSP Problem

```
 1:Initialize the population;
 2:Generate SPTree_s and SPTree_d;
 3:repeat
 4:   if the topology changes then
 5:      Generate SPTree_s and SPTree_d;
 6:      if the number of new wake-up nodes ≠ 0 then
 7:         Generate SPTree_waking;
 8:      end if
 9:      if the number of new sleeping nodes ≠ 0 then
10:         for i ← 1 to popsize do
11:            if individual[i] contains any sleeping node then
12:               v_j ← a node randomly picked before any sleeping node
                  in individual[i];
13:               Generate SPTree_sleeping whose root is v_j;
14:               if SPTree_sleeping contains a node v_k after all sleeping
                  nodes in individual[i] then
15:                  Replace the Path(v_j, v_k, individual[i]) by Path(v_j, v_k,
                     SPTree_sleeping);
16:               else
17:                  Replace the Path(v_j, v_k, individual[i]) by a random-
                     ly generated sub-path from v_j to v_k;
18:               end if
19:            end if
20:         end for
21:      end if
22:   end if
23:   Clear ImpIndSet;
24:   for i ← 1 to popsize do
25:      if individual[i] contains a non-root node v_j in SPTree_s then
26:         ind_new ←Path(s, v_j, SPTree_s) + Path(v_j, d, individual[i]);
27:         Add ind_new into ImpIndSet;
28:      end if
29:      if individual[i] contains a non-root node v_k in SPTree_d then
30:         ind_new ←Path(s, v_k, individual[i]) + Path(v_k, d, SPTree_d);
31:         Add ind_new into ImpIndSet;
32:      end if
33:   end for
34:   if the number of new wake-up nodes ≠ 0 then
35:      for i ← 1 to popsize do
36:         if individual[i] contains two nodes v_j and v_k in SPTree_waking
            then
37:            ind_new ← Path(s, v_j, individual[i]) + Path(v_j, v_k, SPTree_wak-
               ing) + Path(v_k, d, individual[i]);
38:            Add ind_new into ImpIndSet;
39:         end if
40:      end for
41:   end if
42:   Evaluate the population;
43:   Elitism-based immigrants operation;
44:   Selection;
45:   Crossover;
46:   Mutation;
47:   Remove the redundant individuals in the population;
48:   Add the individuals from ImpIndSet into the population;
49:until the termination condition is met
```

In favor of adding the improved individuals to the population and maintaining the diversity of the population, a

strategy of removing redundancies is adopted. That is, the frequency occurrence for each individual is no more than a certain number, which is set to 1 in this paper, namely each individual appears once at most. If there are redundant individuals, the repetitive individuals are eliminated. If *nr* redundant individuals in the population are removed, the removed individuals will be replaced by *nr* improved individuals which are stored in *ImpIndSet*. In case that the total number of individuals in *ImpIndSet* is less than *nr*, a corresponding number of individuals, which are randomly generated, are added into the population to fill the gap. Notice that the individuals in *ImpIndSet* are just suitable for the current population and should be regenerated in the next generation, as is shown in step 23.

## IV. Experiments

In the experiments, the proposed GA for the DSP problems is implemented. The simulation environments proposed in [3] are adopted and the dataset provided in [3] is used for the test. The algorithms with good performance in [3] are picked for comparisons, where the algorithms EIGA, RIGA and HIGA perform well in the acyclic dynamic environments and the algorithms MEGA, MRIGA and MIGA perform well in the cyclic dynamic environments. Therefore, the proposed GA is compared with above GAs in the acyclic dynamic environments and the cyclic dynamic environments, respectively.

### *A. The Simulation of Dynamic Environments*

In this paper, we investigate the network communication problem in the mobile ad hoc network (MANET) [20]. A dynamic simulation environment model for MANET is designed in [3], and the same model is adopted in this paper, which is described as follows. Firstly, a square region with the area of 200 × 200 is specified. Then 100 nodes in the region are randomly generated. If the Euclidean distance between two nodes is not greater than 50, an edge is added to connect them and the cost of this edge is set with a corresponding value. Finally, the topology should be checked to ensure that the topology is connected. If it is not connected, the topology should be regenerated. The nodes in the initial topology are all waking. After a certain number (denoted by *R*) of generations, a certain number (denoted by *M*) of nodes sleep or wake up according to their current status.

The data set provided by [3] is adopted in this paper. The data set includes 4 pieces of data, namely topology series #1, #2, #3 and #4. Topology series #2, #3 and #4 represent the acyclic dynamic environments where *M* equals to 2, 3 and 4 respectively, while topology series #1 represent cyclic dynamic environments. In topology series #1, *M* is set to 2, and topology 1 is the same as topology 21, and the subseries from topology 1 to 21 are repeated five times, and therefore, the cyclic dynamic environments with 101 topologies are created. Besides, for each data pieces, *R* is set to 5, 10 and 15, respectively. For the sake of simplicity and without loss of generality, the source node *s* is set to 1, and the destination node *d* is set to 100.

In order to evaluate the performance of the GAs, the offline performance function is adopted. For each run of an algorithm, the best individual in the population is picked and the total path cost of this individual is calculated. The overall offline performance of a GA is shown as (3) [3].

$$\bar{F}_{OFF} = \frac{1}{G_{max}} \sum_{i=1}^{G_{max}} \left( \frac{1}{N_{run}} \sum_{j=1}^{N_{run}} F_{OFF_{ij}} \right) \qquad (3)$$

In the function, $G_{max}$ represents the total number of evolutionary generations for a run; $N_{run}$ is the total number of runs; $F_{OFF_{ij}}$ is the best fitness in the *i*th generation of the *j*th run. $\bar{F}_{OFF}$ is the offline performance.

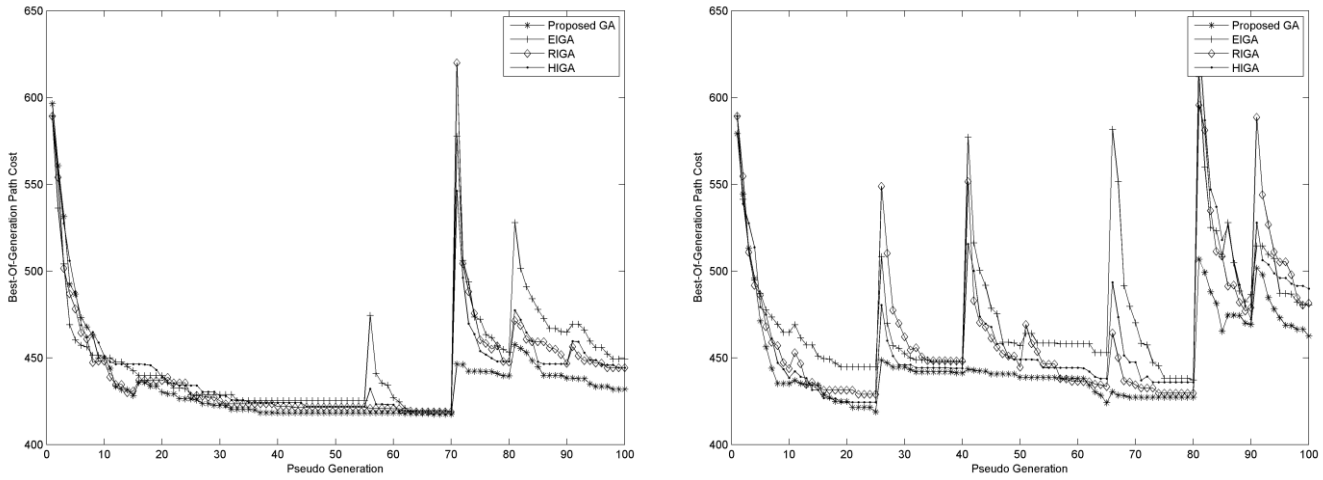
### *B. The Experimental Results and Analyses in the Acyclic Dynamic Environments*

The topology series #2, #3 and #4 are adopted to evaluate the performance of the proposed GA in the acyclic dynamic environments. The GAs (EIGA, RIGA and HIGA) with good performance in the acyclic dynamic environments in [3] are picked out to compare with the proposed GA, and the parameters of the picked GAs are assigned with the same values as mentioned in [3]. For the sake of fairness, the parameters of the proposed GA are assigned with the same values, namely the population size is 50; the mutation probability is 0.1; the parameters for the elitism-based immigrants are the same as those in the EIGA (the percentage of the elitism-based immigrants in the population is 0.2 and the mutation probability for the elite is set to 0.8). Besides, $num_s$, $num_d$, $num_{waking}$ and $num_{sleeping}$ mentioned in section III are all set to 10. In the acyclic dynamic environments, we set *M* to 2, 3 and 4 and set *R* to 5, 10 and 15, and therefore, nine simulation environments are created for each combination. Because there are 21 topologies for each value of *R*, the maximum evolutionary generations for the GAs are 105, 210 and 315, respectively. And the number of runs for the GAs is set to 10.

The first 100 generations of the GAs with *R*=5 over topology series #2 and #4 are picked to present the quality of the solutions in Fig. 4(a) and (b), respectively. From Fig. 4, it is obviously known that the proposed GA achieves the lowest path cost after some generations. The reason is that after some generations, the proposed GA picks out some individuals in *ImpIndSet* to replace the redundant individuals in the population while the other GAs do not. Further, it is known that the proposed GA adapts to the environmental changes rapidly, especially in the generations from 70 to 80 in Fig. 4(a) and from 40 to 50 in Fig. 4(b).

The average optimal path costs over all the generations for each GA in the acyclic dynamic environments are shown in Table I. From Table I, it can be observed that for the same data and the same GA, the average optimal path cost is the maximum in the case with *R* of 5, while the average optimal path cost is the minimum with *R* of 15. This is because the value of *R* represents the generation interval between two environmental changes. Therefore, if the generation interval gets longer, the total evolution times will be longer, and the lower average optimal path cost will be obtained. From Table I, it is known that the proposed GA could get the minimal average optimal path cost, which is shown in bold, compared with the other GAs in all the nine simulation environments.

In order to compare the offline performance of the GAs, we adopt the two-tailed t-test with a 0.05 level of significance.




(a) (b)

Fig. 4. Comparison results of the quality of the solutions for GAs in acyclic dynamic environments over (a) topology series #2 and (b) topology series #4

TABLE I
THE AVERAGE OPTIMAL PATH COSTS OVER ALL THE GENERATIONS IN ACYCLIC DYNAMIC ENVIRONMENTS

| GAs \ Data | Topology Series #2 | | | Topology Series #3 | | | Topology Series #4 | | |
|---|---|---|---|---|---|---|---|---|---|
| | 5 | 10 | 15 | 5 | 10 | 15 | 5 | 10 | 15 |
| EIGA | 448.152 | 436.535 | 436.935 | 460.901 | 451.386 | 442.123 | 474.629 | 456.384 | 452.726 |
| RIGA | 442.614 | 435.407 | 436.278 | 458.702 | 447.658 | 444.445 | 465.824 | 456.996 | 449.575 |
| HIGA | 443.817 | 436.743 | 435.141 | 456.613 | 445.810 | 444.367 | 464.731 | 455.780 | 452.100 |
| **Proposed GA** | **435.104** | **427.242** | **425.213** | **440.581** | **433.868** | **430.615** | **448.477** | **440.919** | **437.174** |

TABLE II
T-TEST RESULTS OF COMPARING GAS IN ACYCLIC DYNAMIC ENVIRONMENTS

| Comparison \ Data | Topology Series #2 | | | Topology Series #3 | | | Topology Series #4 | | |
|---|---|---|---|---|---|---|---|---|---|
| | 5 | 10 | 15 | 5 | 10 | 15 | 5 | 10 | 15 |
| Proposed GA - EIGA | s+ | s+ | s+ | s+ | s+ | s+ | s+ | s+ | s+ |
| Proposed GA - RIGA | s+ | s+ | s+ | s+ | s+ | s+ | s+ | s+ | s+ |
| Proposed GA - HIGA | s+ | s+ | s+ | s+ | s+ | s+ | s+ | s+ | s+ |

And the results are shown in Table II. The sign "s+" means that the GA in the left of "-" is significantly better than the right GA. It is concluded that the proposed GA performs significantly better than all the other GAs in all the acyclic dynamic environments under a 0.05 level of significance.

### *C. The Experimental Results and Analyses in the Cyclic Dynamic Environments*

Because the GAs with memory schemes proposed in [3] are excellent in the cyclic dynamic environments, the proposed GA is compared with the three GAs (MEGA, MRIGA and MIGA) with memory schemes in the cyclic dynamic environments. In the same way, the parameters of the GAs are assigned with the same values specified in [3]. And the parameters of the proposed GA are the same as those mentioned in section IV-B. The topology series #1 are adopted in the cyclic dynamic environments, and $M$ is set to 2. The performances of the GAs are compared in the three environments, in which $R$ is set to 5, 10 and 15 respectively. The topology series #1 contain 101 topologies. Therefore, the maximum generations for the GAs are 505, 1010 and 1515 with different values of $R$, respectively.

We sample the first 100 generations and the generations from 800 to 900 of the GAs with $R$=10 over topology series #1 to present the results in Fig. 5(a) and (b), respectively. From Fig. 5(a), the results show that the proposed GA gets the minimum path cost after some generations. However, the results in Fig. 5(b) show that the quality of the solutions of the proposed GA is worse than those of other GAs sometimes (around generation 810). The reason is that the information in the memory of the memory-based GAs is much too useful due to the cyclically environmental changes.

The average optimal path costs over all the generations for all GAs in the cyclic dynamic environments are shown in Table III. From Table III, compared with the other GAs, the proposed GA could obtain the minimal average optimal path cost which is shown in bold in all the three dynamic environments. However, the differences between the proposed GA's average optimal path cost and the memory-based GAs' in the cyclic environments are not so obvious as those in the cyclic environments.

Similarly, we adopt two-tailed t-test with a 0.05 level of significance to compare the offline performances of the GAs, as shown in Table IV. The signs "+" and "s+" mean that the GA in the left of "-" is better and significantly better than the right GA, respectively. From Table IV, it is known that the proposed GA is significantly better than MEGA and MIGA in all the three environments and significantly better than MRIGA with $R$ of 10. However, the proposed GA is better than but not significantly better than MRIGA when $R$ is set to

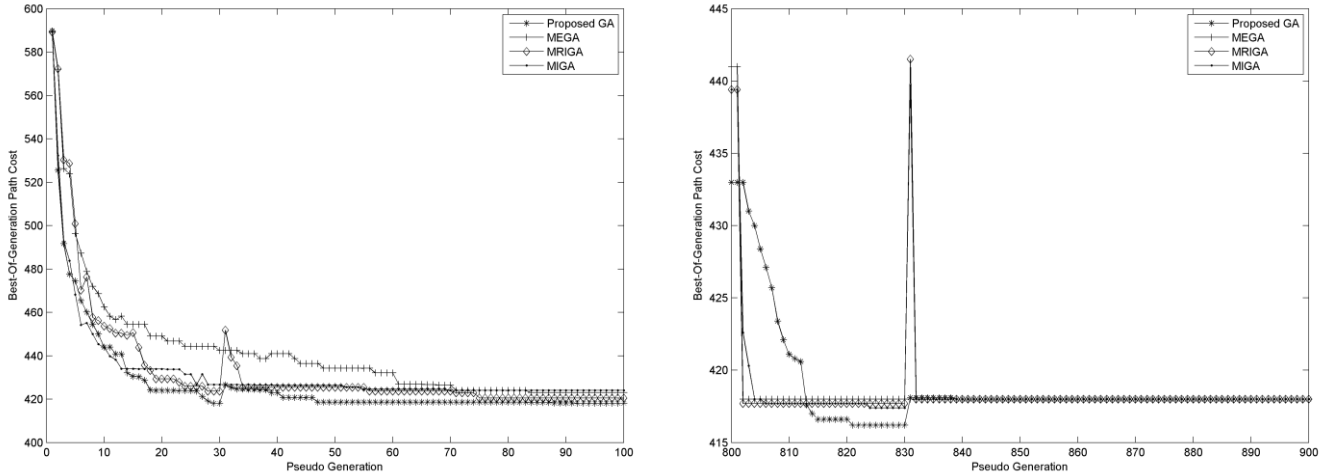



Fig. 5. Comparison results of the quality of the solutions for GAs in cyclic dynamic environments over topology series #1 in the generations (a) from 1 to 100 and (b) from 800 to 900

TABLE III
THE AVERAGE OPTIMAL PATH COSTS OVER ALL THE GENERATIONS IN CYCLIC DYNAMIC ENVIRONMENTS

| GAs \ Data | Topology Series #1 | | |
|---|---|---|---|
| | 5 | 10 | 15 |
| MEGA | 434.242 | 428.675 | 426.731 |
| MRIGA | 429.055 | 427.168 | 425.975 |
| MIGA | 430.863 | 427.759 | 426.189 |
| **Proposed GA** | **426.830** | **424.319** | **423.673** |

TABLE IV
T-TEST RESULTS OF COMPARING GAS IN CYCLIC DYNAMIC ENVIRONMENTS

| Comparison \ Data | Topology Series #1 | | |
|---|---|---|---|
| | 5 | 10 | 15 |
| Proposed GA - MEGA | s+ | s+ | s+ |
| Proposed GA - MRIGA | + | s+ | + |
| Proposed GA - MIGA | s+ | s+ | s+ |

5 and 15. From the results, it is known that the memory-based GAs perform well in the cyclic dynamic environments. If the environments change cyclically, the memory-based GAs could get the highly fit solutions which have been stored in the memory. So that the proposed GA is not significantly better than the memory-based GAs sometimes.

## V. CONCLUSIONS AND FUTURE WORK

The shortest path problems are conventional combinatorial optimization problems. In this paper, an improved GA with four local search operators inspired by Dijkstra's Algorithm is proposed for DSP problems. When the population initializes or the environment changes, the local search operators are carried out to perform local searches around some nodes (namely, the source node, destination node and the nodes affected by the environmental change), and local shortest path trees are generated and stored. Then local shortest path trees are used to enhance the quality of the individuals when the population evolves, so that the proposed GA could get solutions with higher fitness. In order to verify the performance of the proposed GA, we have compared the proposed GA with some other well-behaved GAs in both acyclic and cyclic dynamic environments. The experimental results show that the proposed algorithm could get solutions which adapt to new environments rapidly and produce high-quality solutions after environmental changes.

In the future, we will investigate the DSP problems in the large-scale topologies, and take some missing factors (such as the situation where some nodes move) into consideration. And we will adopt the more ideas from the deterministic algorithms to enhance the performance of GAs. Besides, some dynamic path optimization problems in the real-world applications (such as the car navigation problems) are the multi-objective ones in some situations. Therefore, we will do some work on the multi-objective path optimization problems under dynamic environment.

## REFERENCES


[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik,* vol. 1, pp. 269-271, 1959.

[2] R. Bellman, "On a routing problem," DTIC Document1956.

[3] S. Yang, H. Cheng, and F. Wang, "Genetic Algorithms With Immigrants and Memory Schemes for Dynamic Shortest Path Routing Problems in Mobile Ad Hoc Networks," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on,* vol. 40, pp. 52-63, 2010.

[4] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning,* vol. 3, pp. 95-99, 1988.

[5] T. Back, D. B. Fogel, and Z. Michalewicz, *Handbook of evolutionary computation*: IOP Publishing Ltd., 1997.

[6] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *Computer,* vol. 27, pp. 17-26, 1994.

[7] S. Yang, "Genetic Algorithms with Elitism-Based Immigrants for Changing Optimization Problems," in *Proceedings of the 2007 EvoWorkshops 2007 : Applications of Evolutionary Computing*, 2007, pp. 627-636.

[8] R. Tinós and S. Yang, "A self-organizing random immigrants genetic algorithm for dynamic optimization problems," *Genetic Programming and Evolvable Machines,* vol. 8, pp. 255-286, 2007.

[9] S. Yang and R. Tinós, "A hybrid immigrants scheme for genetic algorithms in dynamic environments," *International Journal of Automation and Computing,* vol. 4, pp. 243-254, 2007.

[10] S. Yang, "Memory-based immigrants for genetic algorithms in dynamic environments," in *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, 2005, pp. 1115-1122.
[11] H. Cheng and S. Yang, "Genetic algorithms with immigrants schemes for dynamic multicast problems in mobile ad hoc networks," *Engineering Applications of Artificial Intelligence,* vol. 23, pp. 806-819, 2010.
[12] S. Yang, "Genetic algorithms with memory- and elitism-based immigrants in dynamic environments," *Evolutionary Computation,* vol. 16, pp. 385-416, 2008.
[13] C. W. Ahn and R. S. Ramakrishna, "A genetic algorithm for shortest path routing problem and the sizing of populations," *Evolutionary Computation, IEEE Transactions on,* vol. 6, pp. 566-579, 2002.
[14] M. Gen, R. Cheng, and D. Wang, "Genetic algorithms for solving shortest path problems," in *Evolutionary Computation, IEEE International Conference on*, 1997, pp. 401-406.
[15] M. Gen and L. Lin, "A new approach for shortest path routing problem by random key-based GA," in *Proceedings of the 2006 Conference on Genetic and Evolutionary Computation*, 2006, pp. 1411-1412.
[16] H. Cheng and S. Yang, "Genetic algorithms with elitism-based immigrants for dynamic shortest path problem in mobile ad hoc networks," in *Evolutionary Computation, IEEE Congress on*, 2009, pp. 3135-3140.
[17] H. Cheng and S. Yang, "Multi-population genetic algorithms with immigrants scheme for dynamic shortest path routing problems in mobile ad hoc networks," in *Proceedings of the 2010 International Conference on Applications of Evolutionary Computation-Volume Part I*, 2010, pp. 562-571.
[18] S. Lee, S. Soak, K. Kim, H. Park, and M. Jeon, "Statistical properties analysis of real world tournament selection in genetic algorithms," *Applied Intelligence,* vol. 28, pp. 195-205, 2008.
[19] R. Poli and W. B. Langdon, "Schema theory for genetic programming with one-point crossover and point mutation," *Evolutionary Computation,* vol. 6, pp. 231-252, 1998.
[20] S. Corson and J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations," *Network Working Group,* 1999.