Chapter 13
Ant Colony Optimization Algorithms with Immigrants Schemes for the Dynamic Travelling Salesman Problem

Michalis Mavrovouniotis and Shengxiang Yang

Abstract. Ant colony optimization (ACO) algorithms have proved to be powerful methods to address dynamic optimization problems (DOPs). However, once the population converges to a solution and a dynamic change occurs, it is difficult for the population to adapt to the new environment since high levels of pheromone will be generated to a single trail and force the ants to follow it even after a dynamic change. A good solution is to maintain the diversity via transferring knowledge from previous environments to the pheromone trails using immigrants. In this chapter, we investigate ACO algorithms with different immigrants schemes for two types of dynamic travelling salesman problems (DTSPs) with traffic factor, i.e., under random and cyclic dynamic changes. The experimental results based on different DTSP test cases show that the investigated algorithms outperform other peer ACO algorithms and that different immigrants schemes are beneficial on different environmental cases

13.1 Introduction

Ant colony optimization (ACO) algorithms are inspired from the behaviour of real ant colonies when they search for food from their nest to food sources. A colony of ants communicates via the pheromone trails in order to complete their food-searching task as efficiently as possible. ACO algorithms have proved that they are good meta-heuristics to many difficult optimization problems [11, 12, 15, 36].

The first optimization problem addressed by ACO algorithms was the travelling salesman problem (TSP), where a population of ants is placed on each city randomly and walk to the edges of the cities until each ant generates a feasible tour, in which all customers are satisfied [13]. Each ant writes pheromone to the trail of its tour for the other ants to read it while they construct their tours.

Michalis Mavrovouniotis · Shengxiang Yang
Centre for Computational Intelligence (CCI), School of Computer Science and Informatics, De Montfort University, The Gateway, Leicester LE1 9BH, U.K.
e-mail: {mmavrovouniotis,syang}@dmu.ac.uk

S. Yang and X. Yao (Eds.): *Evolutionary Computation for DOPs*, SCI 490, pp. 317–341.
DOI: 10.1007/978-3-642-38416-5_13

Researchers have mainly focused on ACO for stationary optimization problems (SOPs), where the environment remains fixed during the execution of the algorithm [2, 3, 28]. However, in many real-world applications we have to deal with dynamic optimization problems (DOPs), where the problem, including the objective function, the variables, the problem instance, the constraints, and so on, may change over time [26]. Usually, such uncertainties cause the optimum to move. For example, a dynamic version of the TSP can be generated where the cost of the edges between two cities may increased, representing potential traffic jams. The objective of the dynamic TSP (DTSP) is not only to converge and output a near optimum (or the optimum) solution quickly, as in the static TSP, but to also track and output the moving optimum.

Considering the DTSP, traditional ACO algorithms may face a serious challenge due to the fact that the pheromone trails of the previous environment will not be compatible with the new environment when a dynamic change occurs. A simple way to address this problem is to re-initialize the pheromone trails with an equal amount and consider every dynamic change as the arrival of a new problem that needs to be solved from scratch. This strategy acts as a restart of the algorithm which is computationally expensive and usually not efficient. Moreover, in order to perform this action, the dynamic change needs to be detected which usually is not possible on DOPs [31].

However, it is believed that ACO algorithms can adapt to DOPs since they are inspired from nature which is a continuous adaptation process [5, 26]. Since ACO algorithms have been designed for SOPs lose their adaptation capabilities quickly because of stagnation behaviour, where all ants follow the same path from the early stages of the execution. Recently, several approaches have been proposed to avoid stagnation behaviour and address DTSPs, which includes: (1) local and global restart strategies [21]; (2) pheromone manipulation schemes to maintain diversity [16]; (3) increase diversity via immigrants schemes [29, 31]; (4) memory-based approaches [19, 22]; (5) and memetic algorithms [30].

Among these approaches, immigrants schemes have been found beneficial when integrated with ACO algorithms for different DTSPs. Every iteration, immigrant ants are generated and replace a small portion of the worst ants in the current population. This action is performed before pheromone is updated, in order to bias the ants of the next iteration with the diversity and knowledge transferred from the immigrant ants. Immigrants schemes mainly differ on the way immigrant ants are generated.

In this chapter, all different ACO algorithms based on immigrants schemes are examined intensively, and compared with other peer ACO algorithms for different DTSPs cases. The contents of this chapter are categorized as follows. Section 13.2 describes the DTSPs used in the experiments. Section 13.3 describes traditional ACO algorithms for the DTSP, whereas Section 13.4 gives details of the investigated ACO algorithms based on immigrants schemes. Section 13.5 presents the experimental results and analysis. Finally, Section 13.6 concludes this contribution and points out future work.

13.2 Dynamic Travelling Salesman Problem with Traffic Factor

The TSP is the most fundamental, popular and well-studied *NP*-hard combinatorial optimization problem. It can be described as follows: Given a collection of cities, we need to find the shortest path that starts from one city and visits each of the other cities once and only once before returning to the starting city. Usually, the problem is represented by a fully connected weighted graph $G = (V, E)$, where V is a set of n vertices and E is a set of edges. The collection of cities is represented by the set V and the connections between them by the set E. Each connection is associated with a cost D_{ij}, which represents the distance (or travel time) between cities i and j.

Many algorithms, either exact algorithms or approximation algorithms, including ACO have been proposed to solve the static TSP [13, 27, 33]. Although exact algorithms guarantee to provide the global optimum solution, in the case of *NP*-hard problems, they need, in the worst case, exponential time to find it. On the other hand, approximation algorithms can provide a solution efficiently but cannot guarantee the global optimum [24, 35].

The TSP becomes more challenging and realistic if it is subject to a dynamic environment. For example, a salesman wants to distribute items sold in different cities starting from his home city and returning after he visited all the cities to his home city again. The task is to optimize his time and plan his tour as efficiently as possible. Therefore, by considering the distances between cities it can generate the route and start the tour. However, it is difficult to consider traffic delays that may affect the route. Traffic delays may change the time planned beforehand, and the salesman will need a new alternative route fast to avoid long traffic delays and optimize his time again.

There are several variations of DTSPs considered in the literature, such as changing the topology of cities by replacing cities [19, 21, 29, 30], and changing the distances between cities by adding traffic factors to the links between cities [16, 31, 32]. In DTSPs where cities are replaced, each city has a probability m to be replaced regularly in time, usually measured in a certain number of iterations of running an algorithm. On the other hand, in DTSPs with traffic factors, each link has a probability m to add or deduce traffic regularly in time.

13.2.1 DTSP with Random Traffic

In this chapter, we generate DTSPs with traffic factors as follows. We assume that the cost of the link between cities i and j is $D_{ij} = D_{ij} \times F_{ij}$, where D_{ij} is the normal travelled distance and F_{ij} is the traffic factor between cities i and j. Every f iterations of running an algorithm, a random number in $[F_L, F_U]$ is generated probabilistically to represent the traffic factor between cities, where F_L and F_U are the lower and upper bounds of the traffic factor, respectively. Each link has a probability m to add traffic every f iteration, where the traffic factor F_{ij} of the remaining links is set to 1, which indicates no traffic.

For example, a dynamic case with high traffic is constructed by setting traffic factor values closer to F_U with a higher probability to be generated, while for a

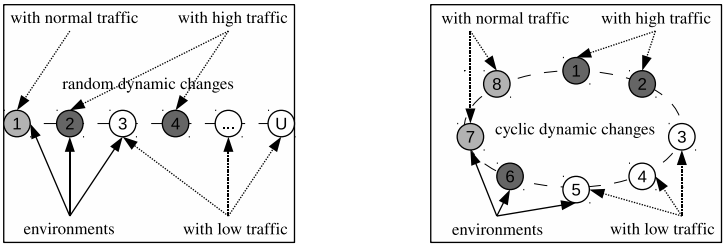

Fig. 13.1 Illustration of a random dynamic environment with unlimited states and a cyclic dynamic environment with 8 states. Each node represents a different environment where white, light grey, and dark grey, represent low, medium, and high traffic jams, respectively

dynamic case with low traffic, a higher probability is given to traffic factor values closer to F_L. This type of environments are denoted *random DTSPs* in this chapter because previously visited environments are not guaranteed to reappear.

13.2.2 DTSP with Cyclic Traffic

Another variation of the DTSP with traffic factors is the DTSP where the dynamic changes occur with a cyclic pattern. In other words, previous environments will appear again in the future. Such environments are more realistic since they represent a 24-hour traffic jam situation in a day.

A cyclic environment can be constructed by generating different dynamic cases with traffic factors as the base states, representing DTSP environments where each link has a probability m to add low, normal, or high traffic as in random DTSPs. Then, the environment cycles among these base states, every f iteration, in a fixed logical ring as in Fig. 13.1. Depending on the period of the day, dynamic cases with different traffic factors can be generated. For example, during the rush hour periods, a higher probability is given to the traffic factors closer to F_U, whereas during evening hour periods, a lower probability is given to F_U and a higher probability to F_L. This type of environments are denoted as *cyclic DTSPs* in this chapter because previously visited environments will reappear several times.

13.3 Ant Colony Optimization for the DTSP

ACO consists of a population of μ ants that construct solutions, i.e., tours in the TSP, and update their trails with pheromone according to the solution quality [9]. Considering the TSP, the ants "walk" on the links between the cities, where they "read" pheromone from the links or "write" additional pheromone to the links.

Initially, all trails are assigned with an equal amount of pheromone, i.e., τ_{init}, and each ant is placed on a randomly selected city. With a probability $1 - q_0$, where $0 \leq q_0 \leq 1$ is a parameter of the decision rule, an ant k chooses the next city j from city i, probabilistically, as follows:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}, \text{if } j \in N_i^k, \tag{13.1}$$

where τ_{ij} and $\eta_{ij} = 1/D_{ij}$ is the existing pheromone trail and heuristic information available a priori, respectively, where D_{ij} is the cost between cities i and j (including the traffic factor). N_i^k denotes the neighbourhood of cities of ant k that have not yet been visited when its current city is i. α and β are the two parameters that determine the relative influence of the pheromone trail and heuristic information, respectively. With the probability q_0, ant k chooses the next city, i.e, z, with the maximum probability which satisfies the following formula:

$$z = \underset{j \in N_i^k}{\text{argmax}} \; [\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}. \tag{13.2}$$

This process continues until each ant has visited all cities once. Thereafter, the ants update their pheromone trails. The different variations of ACO algorithms mainly differ in the way pheromone trails are updated [4, 10, 13, 14, 37].

13.3.1 Standard ACO

The state-of-the-art ACO on the static TSP is the MAX-MIN ant system (MMAS) [38]. In MMAS, the ants construct solutions using Eq. (13.1) and only the best ants are allowed to retrace their solution and deposit pheromone as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best}, \forall\,(i,j) \in T^{best}, \tag{13.3}$$

where T^{best} is the tour of the best ant and $\Delta\tau_{ij}^{best} = 1/C^{best}$, where C^{best} is the cost of the tour T^{best}. However, the ant allowed to deposit pheromone may be either the best-so-far ant, in which case $\Delta\tau_{ij}^{best} = 1/C^{bs}$, where C^{bs} is the tour cost of the best-so-far ant, or the iteration-best ant, in which case $\Delta\tau_{ij}^{best} = 1/C^{ib}$, where C^{ib} is the tour cost of the best ant of the iteration. Both update rules are used in an alternative way under a pre-defined criteria (for more details see [38]).

In addition, a constant amount of pheromone is deducted from all trails due to the pheromone evaporation, which is defined as:

$$\tau_{ij} \leftarrow (1-\rho)\,\tau_{ij}, \forall\,(i,j), \tag{13.4}$$

where $0 < \rho \leq 1$ is the rate of evaporation.

Moreover, the pheromone trail values in MMAS are kept to the interval $[\tau_{min}, \tau_{max}]$ and they are re-initialized to τ_{max} every time the algorithm shows a stagnation behaviour, where all ants follow the same path or when no improved tour has been found for several iterations. The MMAS, is denoted as S-ACO, and it is used in the experimental study later in this chapter.

13.3.2 Population-Based ACO (P-ACO)

The P-ACO algorithm was first applied on the static TSP [20]. Later on, it has been applied to the DTSP where a small portion of cities is replaced by other new ones [19, 22]. In P-ACO, ants construct solutions using Eq. (13.1). However, it differs from the S-ACO, since it maintains a population-list (memory) of ants (solutions), denoted k_{long} of limited size K_l, and stores the iteration-best ant in every iteration.

The pheromone update depends on k_{long}, where every time the iteration-best ant enters k_{long}, a positive constant update is added to its corresponding pheromone trails, which is defined as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{ib}, \forall\ (i,j) \in T^{ib}, \tag{13.5}$$

where $\Delta\tau_{ij}^{ib} = (\tau_{max} - \tau_{init})/K_l$ and T^{ib} is the tour of the iteration-best ant. Moreover, τ_{max} and τ_{init} denote the maximum and initial pheromone amount, respectively. When k_{long} is full, the iteration-best ant needs to replace an ant k stored in k_{long}, and, thus, a negative constant update to its corresponding pheromone trails is done, which is defined as follows:

$$\tau_{ij} \leftarrow \tau_{ij} - \Delta\tau_{ij}^{k}, \forall\ (i,j) \in T^{k}, \tag{13.6}$$

where $\Delta\tau_{ij}^{k}$ is defined as in Eq. (13.5) and T^k is the tour of the ant to be replaced. Pheromone evaporation is not used in the P-ACO algorithm.

Several strategies regarding which ant should the iteration-best ant replace in k_{long} have been proposed, such as *Age*, *Prob*, and *Quality* [19]. In the default strategy, i.e., *Age*, the iteration-best ant replaces the ant which has entered k_{long} first. In the *Prob* strategy, the iteration-best ant can replace any ant probabilistically, and in the *Quality* strategy, the worst ant is replaced. Experiments show that the *Age* strategy is more consistent and performs better than the others, since other strategies have more chances to maintain identical ants into k_{long}, which leads the algorithm to the stagnation behaviour [19]. This is due to the fact that high levels of pheromone will be generated into a single trail and dominate the search space. Therefore, the P-ACO algorithm with the *Age* strategy is used in the experimental study later in this chapter.

13.3.3 React to Dynamic Changes

In Bonabeau *et al.* [5], it was discussed that traditional ACO algorithms may have good performance for DTSPs, since they are very robust algorithms. The mechanism which enables ACO algorithms to adapt to DOPs is the pheromone evaporation. Lowering the pheromone values enables the algorithm to forget bad decisions made in previous iterations. Moreover, when a dynamic change occurs, it will eliminate the pheromone trails of the previous environment that are not useful in the new environment, where the ants may be biased and not adapt well.

The S-ACO algorithm can be applied directly to the proposed DTSPs with traffic factors, either random or cyclic, without any modifications, apart from the heuristic

information where the traffic factor needs to be considered. Further special measures when a dynamic change occurs are not required.

Similar to S-ACO, P-ACO can be applied directly to the proposed DTSPs. The ants stored in the k_{long} are re-evaluated in every iteration to be consistent with the changing environments.

13.4 Investigated ACO Algorithms with Immigrants Schemes

ACO algorithms are constructive heuristics, where every iteration ants move from one city to the next city probabilistically, until they generate feasible solutions as described in Section 13.3. At the end of each iteration the constructed solutions are cleared to generate new ones, but every ant deposits pheromone and leave a trail to the corresponding solutions, e.g., the links between the cities of a TSP tour as in Eq. 13.3. In contrast, genetic algorithms (GAs) are based on a pre-defined set of feasible solutions (population of individuals), e.g., a set of tours for TSP. The population is directly transferred from one iteration to the next using the fittest individuals [25, 33]. Search operators, i.e., crossover and mutation, are used to generate the new population of solutions, which is usually better than the previous one.

The P-ACO framework is based on ants that construct solutions, where the best ant of each iteration is stored in an actual population as in a GA and they are transferred directly to the next iteration. The solutions in the population are then used to update the pheromone information for the new ants of the new iteration. The population-list is updated every iteration as described in Section 13.3.

13.4.1 General Framework of ACO with Immigrants Schemes

The framework of ACO algorithms with immigrants schemes is inspired from the GA characteristics of the P-ACO framework and the good performance of immigrants schemes in GAs for binary-encoded DOPs [39, 40, 43, 45]. Considering that P-ACO maintains a population of solutions, immigrant ants can be generated and replace ants in the current population. The aim of the proposed framework is to maintain the diversity within the population and transfer knowledge from previous environments to the pheromone trails of the new environment.

The main idea is to generate the pheromone information for every iteration considering information from the pheromone trails of the previous environment and extra information from the immigrant ants generated. Therefore, instead of using a long-term memory k_{long} as in P-ACO, a short-term memory is used, denoted k_{short}, where all ants stored from iteration $t-1$ are replaced by the first K_s best ants of the current iteration t, where K_s is the size of k_{short}, instead of only replacing the oldest one as in P-ACO. Moreover, immigrant ants are generated and replace the worst ants in k_{short} with the replacement rate r, usually small. Therefore, when ants are removed, a negative update is made to their pheromone trails as in Eq. (13.6), and when new ants are added, a positive update is made to their pheromone trails as in Eq. (13.5). This process is repeated as represented in Fig. 13.2.

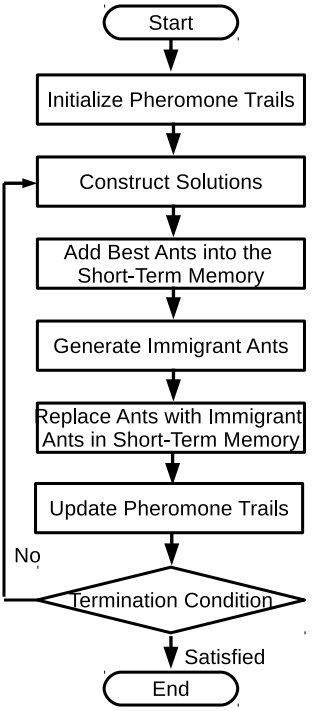

Fig. 13.2 General framework of ACO algorithms with immigrants schemes

The benefits of using k_{short} are closely related to the survival of ants in a dynamic environment, where no ant can survive in more than one iteration. For example, in iteration t, if ants are stored from iteration $t-2$ and an environmental change occurred in iteration $t-1$, then the solutions may not be feasible for the current environment in iteration t, and hence need to be repaired. Usually, a repair procedure is computationally expensive, and requires prior knowledge of the problem. Furthermore, this action can be taken only if the environmental changes can be detected, which is usually not applicable in real-world applications. As discussed previously, the S-ACO algorithm with a re-initialization of pheromone trails may not be a sufficient choice on DOPs where the frequency of change is not available beforehand.

The investigated algorithms follow the framework described above, but they differ on the way immigrant ants are generated. The algorithms have been applied on different DTSPs as follows. Random immigrants ACO (RIACO), elitism-based immigrants ACO (EIACO) and hybrid immigrants ACO (HIACO) were applied on a DTSP were cities are added/removed [29]. Memory-based immigrants ACO (MIACO) were applied on a DTSP with cyclic traffic factor as described in Section 13.2 [31]. Environmental-information immigrants ACO (EIIACO) were applied on the DTSP with random traffic factor as described in Section 13.2 [32]. In this chapter,

we re-investigate and compare all the algorithms on the same DTSPs, i.e., on both DTSP with random and cyclic traffic factor.

13.4.2 ACO with Random Immigrants

The traditional random immigrants have been found beneficial for ACO for the DTSP, since they maintain a certain level of diversity during the execution [29]. The principle is to introduce new randomly generated immigrant ants to the population. Therefore, before the pheromone trails are updated, a set S_{ri} of $r \times K_s$ immigrants are randomly generated to replace the worst ants in k_{short}, where r is the replacement rate and K_s is the size of k_{short}.

The RIACO algorithm was proposed to address DTSPs with significantly changing environments. This is because it was claimed that the adaptation of ACO algorithms makes sense only when the environmental changes of a problem are small to medium [6, 26]. This is due to the fact that a new environment has more chance to be similar with the old one. After a change occurs, transferring knowledge from the old environment to the pheromone trails may move the ants into promising areas in the new environment.

Considering the above argument, when the changing environments are not similar or when their is not enough time to gain knowledge from the previous environment, i.e., fast changing environment, the knowledge transferred may misguide the ants from tracking the optimum. Therefore, in such environmental cases is better to generate random diversity, instead of guided diversity by transferring knowledge. However, there is a high risk of randomization, if too much diversity is generated from the immigrants.

13.4.3 ACO with Elitism-Based Immigrants

Differently from RIACO, which generates diversity randomly, EIACO generates guided diversity by transferring knowledge from previous environments and it was proposed to address DTSPs with slowly and slightly changing environments [29]. For each iteration t, within EIACO, the elite from the previous environment, i.e., the best ant from $k_{short}(t-1)$, is used as the base to generate a set S_{ei} of $r \times K_s$ elitism-based immigrants, where r is the replacement rate and K_s is the size of the k_{short} memory.

An elitism-based immigrant is generated using the inversion operator based on the inver-over operator as follows [23]. First, one city, i.e., c, is selected randomly from the best ant of $k_{short}(t-1)$; then with probability p (usually 0.02) the second city c' is selected from $k_{short}(t-1)$; otherwise another ant from μ is randomly selected and assign as the second city c' the next city to the city c. The segment from the next city of c to city c' is reversed and c is set to c'. This process continues until the selected second city c' appears next or previous to the first city c. From the resulting tour an elitism-based immigrant is generated, which inherits some segments

from the elite of the previous environment and some random segments from other ants.

The EIACO algorithm is beneficial in cases where the changing environments are similar, e.g., slightly changing environments, and when the population has sufficient time to converge into a good solution and gain knowledge in the previous environment, e.g., slowly changing environments. Transferring the knowledge gained from the previous environment, to the pheromone trails of the new environment will make sense and guide the population of ants to promising areas.

However, if too much information is transferred, the run basically starts near a local optimum, and get stuck there. Therefore, in some cases with slightly changing environments, EIACO may not perform well. On the contrast, RIACO may generate high level of diversity in slightly changing environments, and degrade the performance of ACO.

13.4.4 ACO with Hybrid Immigrants

The HIACO algorithm uses an immigrants scheme that combines both random and elitism-based immigrants [29]. For each iteration t within HIACO, a set $S_{hi} = S_{ri} + S_{ei}$ hybrid immigrants are generated, where S_{ri} and S_{ei} are two sets of $(r \times K_s)/2$ random and elitism-based immigrants, respectively, r is the replacement rate and K_s is the size of the k_{short} memory. HIACO attempts to combine the merits of both RIACO and EIACO, where one is good on slowly and slightly changing environments and the other on fast and significantly changing environments.

Considering the fact that RIACO face the risk of randomization because of too much diversity, and the fact the EIACO face the risk of too much transferred knowledge, the HIACO may promote the performance of both algorithms. The two types of immigrants may cooperate to address all cases of dynamic environments. For example, in cases the random immigrant will generate high levels of diversity, the elitism-based immigrants will decrease the levels of diversity. On the other hand, if too much knowledge is transferred from elitism-based immigrants and the population gets trapped in local optimum, the random immigrants will help the population to escape from it.

13.4.5 ACO with Memory-Based Immigrants

Differently from EIACO, where the best ant from the previous environment is used as the base to generate immigrants, MIACO uses the best ant from several environments as the base to generate immigrants [31]. The only difference between MIACO and EIACO lies in that MIACO uses both k_{short} and k_{long}, where the first type of memory is updated and used as in RIACO and EIACO. The second type of memory is initialized with random ants and updated by replacing any of the randomly initialized ants if they still exists in the memory, with the best-so-far ant; otherwise, the closest ant in the memory is replaced with the best-so-far ant if it is better. Note that the update strategy of k_{long} in MIACO is different from P-ACO regarding which ant

to replace, since in MIACO the most similar memory updating strategy is used [6], whereas in P-ACO, the new ant replaces the oldest one. In MIACO, a metric of how close ant i is to ant j is used and defined as follows:

$$M_{ij} = 1 - \frac{CE_{ij}}{n}, \tag{13.7}$$

where CE_{ij} is defined as the number of common edges between ant i and ant j, and n is the number of cities. A value M_{ij} closer to 0 means that the ants are closer since they are more similar [1].

Apart from which ant is replaced in k_{long}, the update strategy of MIACO is different from the one used in P-ACO with respect to when an ant is replaced. In P-ACO, the update occurs every iteration, whereas in MIACO the update occurs whenever a dynamic change is detected in order to store useful solutions from different environments.

For each iteration within MIACO, the ants in k_{long} are re-evaluated in order to be valid with the new environment and to detect an environmental change. An environmental change is detected if there is a change in the total cost of ants currently stored in k_{long}. Then, the best ant from k_{long} is selected and used as the base to generate a set S_{mi} of $r \times K_s$ memory-based immigrants, where r is the replacement rate and K_s is the size of the k_{short} memory. A memory-based immigrant is generated using the inver-over operator as in EIACO, but instead of selecting the best ant from $k_{short}(t-1)$, the best ant from $k_{long}(t)$ is selected.

MIACO inherits the advantages of the memory scheme to guide the population directly to an old environment already visited and maintains diversity with immigrants in order to avoid the stagnation behaviour of ACO algorithms. It is very important to store different solutions in k_{long} which represent good solutions for the different environments that may be useful in the future. The key idea behind MIACO is to provide guided diversity into the pheromone trails in order to avoid the disruption of the optimization process [41].

MIACO may be beneficial on the same environmental cases with EIACO since it is a generalized version of EIACO. However, it may be also advantageous in cases where the previous environments will reappear in the future, e.g., cyclic DTSPs.

13.4.6 ACO with Environmental-Information Immigrants

The information obtained from EIACO and MIACO to transfer knowledge is based on individual information, i.e., the best ant from k_{short} and k_{long}, respectively. The EIIACO algorithm generates immigrants using environmental information, i.e., a population of best ants, to transfer knowledge from the previous environment to the new one, in order to address slowly and slightly changing environments [32]. The knowledge transferred from EIIACO contains much more information than the EIACO and MIACO algorithms. EIIACO follows the same framework with other ACO algorithms based on immigrants schemes.

Environmental information-based immigrants are generated using all the ants stored in k_{short} of the previous environment and replace the worst ants in the current k_{short}. Within EIIACO, a probabilistic distribution based on the frequency of cities is extracted, representing information of the previous environment, and is used as the base to generate immigrant ants. The frequency vector of each city i, i.e, $\mathbf{D}_{c_i}$, is constructed by taking the ants of k_{short} as a dataset and locating city c_i from them. The successor and predecessor cities, i.e., c_{i-1} and c_{i+1}, respectively, of city c_i are obtained and update $\mathbf{D}_{c_i}$ accordingly. Note that both cities are recorded since the TSP solution is cyclic. For example, one is added to the corresponding position $i-1$ and $i+1$ in $\mathbf{D}_{c_i}$. The process is repeated for all cities and a table $S = (\mathbf{D}_{c_1}, \ldots, \mathbf{D}_{c_n})$ is generated (where n is the number of cities) .

An environmental information-based immigrant ant, i.e., $A_{eii} = (c_1, \ldots, c_n)$, is generated as follows. First, randomly select the start city c_1; then, the probabilistic distribution of $\mathbf{D}_{c_{i-1}} = (d_1, \ldots, d_n)$ is used to select the next city c_i probabilistically as follows:

$$p_i = \frac{d_i}{\sum_{j \in \mathbf{D}_{c_{i-1}}} d_j}, \text{if } i \in \mathbf{D}_{c_{i-1}}, \tag{13.8}$$

where d_i is the frequency number where city c_i appears before or after city c_{i-1}. Note that all cities currently selected and stored in A_{eii} have a probability of 0.0 to be selected since they are already visited. In the case where the sum of $p_i = 0.0$, which means that all cities in $\mathbf{D}_{c_i}$ are visited, a random city j that has not been visited yet is selected. This probabilistic selection is repeated until all cities are used in order to generate a valid immigrant ant based on the environmental information.

13.5 Experiments

13.5.1 Experimental Setup

The investigated algorithms were tested on the DTSP instances that are constructed from three static benchmark TSP instances taken from TSPLIB[1], i.e., `pr76`, `pr152`, `pr264`, indicating small, medium, and large scale problem instances in this chapter, respectively, in order to investigate the effect of the corresponding immigrants schemes on ACO algorithms for the DTSP.

Our implementation follows the guidelines of the ACOTSP[2] application. Using the methods described in Section 13.2, we have generated two kinds of DTSPs, with random and cyclic traffic factors, respectively, with $F_L = 0$ and $F_U = 5$. For cyclic DTSP, four cyclic states are used. For both types of DTSPs, the value of f was set to 5 and 50, indicating fast and slow environmental changes, respectively. The value of m was set to 0.1, 0.25, 0.5, and 0.75, indicating the degree of environmental changes from small, to medium, and large, respectively. As a result, eight dynamic test DTSPs, i.e., two values of f $\times$ four values of m, were generated from each

[1] See `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`
[2] See `http://www.aco-metaheuristic.org/aco-code`

static TSP instance. Therefore, in order to systematically analyse the adaptation and searching capabilities of each algorithm on the DTSP, 24 dynamic test cases are used, i.e., three problem instances $\times$ eight cases each, for each type of DTSP, i.e., with random and cyclic traffic factors.

For each algorithm on a DTSP, 30 independent runs were executed on the same environmental changes. The algorithms were executed for 1000 iterations and one observation was taken on each iteration. The overall performance of an algorithm on a DTSP instance is defined as follows:

$$\bar{P}^{best} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N} P_{ij}^{best}\right), \tag{13.9}$$

where G is the total number of iterations, N is the number of runs and P_{ij}^{best} is the tour cost of the best-so-far ant, after a change, of iteration i of run j, respectively.

13.5.2 Parameter Settings

The parameters of the investigated algorithms are chosen from our preliminary experiments and some of them are taken from the literature [29, 31]. For all algorithms $\alpha = 1$, $\beta = 5$, $q_0 = 0.0$, $r = 0.3$, $K_s = 10$, and for MIACO $K_l = 4$.

The population of ants μ for each algorithm varies in order to have the same number of evaluations every iteration, i.e., 25. The population of RIACO and EI-IACO was set to $\mu = 25$, for EIACO and HIACO was set to $\mu = 24$ and for MIACO was set to $\mu = 21$. This is because MIACO has a k_{long} memory of size K_l where the solutions need to be re-evaluated on every change to detect dynamic changes, and, thus, $\mu = \mu - K_l$, whereas EIACO and HIACO re-evaluates the best ant from the previous iteration, which counts as a single evaluation, and, thus, $\mu = \mu - 1$. The ants in the k_{short} memory, including the generated immigrants, do not count as evaluations since they are removed every iteration. Moreover, the pheromone they deposit is not based on the quality of the solution as in the S-ACO. Instead, it is a constant value as in P-ACO.

13.5.3 Experimental Results and Analysis of the Investigated Algorithms

The experimental results regarding the offline performance of the investigated algorithms in both DTSPs with random and cyclic traffic factors are presented in Tables 13.1 and 13.2, respectively. The corresponding two-tailed t-test results with 58 degrees of freedom at a 0.05 level of significance are presented in Table 13.3. In the comparisons, "+" or "−" indicates that the first algorithm is significantly better or the second algorithm is significantly better, respectively, and "∼" indicates no significance between the algorithms. Moreover, to better understand the dynamic behaviour of algorithms, the offline performance against the first 500 iterations is plotted in Fig. 13.3 for random DTSPs for $f = 50$ and $m = 0.10$ and $m = 0.75$,

Table 13.1 Experimental results of algorithms regarding the offline performance for random DTSPs

Alg. & Inst.	pr76							
	$f = 5$				$f = 50$			
$m \Rightarrow$	0.10	0.25	0.50	0.75	0.10	0.25	0.50	0.75
RIACO	53405.77	61545.11	78939.34	109341.65	50484.60	56943.36	71997.99	95742.96
EIACO	53481.98	61995.73	79939.40	111215.18	50740.69	57488.74	72295.83	96720.11
HIACO	53299.32	61708.22	79278.76	109990.85	50282.14	56778.05	71587.04	95796.19
MIACO	53669.59	62367.39	80503.22	111982.58	50786.29	57582.64	72551.06	96996.51
EIIACO	53856.99	62156.46	79909.29	110854.31	50745.09	57363.72	72405.07	96330.06
Alg. & Inst.	pr152							
	$f = 5$				$f = 50$			
$m \Rightarrow$	0.10	0.25	0.50	0.75	0.10	0.25	0.50	0.75
RIACO	43731.12	49856.78	61702.52	84089.01	40606.23	45251.32	55209.46	73016.88
EIACO	43582.79	49869.53	62185.09	84449.34	41013.54	45154.71	54581.22	71855.33
HIACO	43579.22	49922.58	61935.99	84262.97	40561.09	44913.56	54486.51	71830.50
MIACO	43777.26	50226.28	62633.86	85252.96	41047.21	45387.66	54854.02	72307.76
EIIACO	43975.13	50167.45	62216.30	84511.91	41087.96	45643.79	55308.69	72920.66
Alg. & Inst.	pr264							
	$f = 5$				$f = 50$			
$m \Rightarrow$	0.10	0.25	0.50	0.75	0.10	0.25	0.50	0.75
RIACO	27803.32	32317.14	41440.08	56978.59	25707.36	29019.14	36155.71	48289.91
EIACO	27631.00	32284.07	41686.12	57496.55	25636.91	28906.24	35834.23	47431.27
HIACO	27801.55	32452.96	41686.40	57321.89	25547.00	28849.66	35877.25	47629.74
MIACO	27720.96	32463.54	41914.54	57863.13	25697.97	29022.73	36038.26	47866.07
EIIACO	27757.27	32338.73	41637.13	57420.54	25788.50	29080.56	36237.60	48362.34

and the offline performance against the first 100 iterations is plotted in Fig. 13.4 for cyclic DTSPs for $f = 5$ and $m = 0.10$ and $m = 0.75$. From the experimental results, several observations can be made by comparing the behaviour of the algorithms.

First, RIACO outperforms EIACO, MIACO, EIIACO in almost all dynamic cases with $f = 5$ and $m = 0.75$ in both random and cyclic DTSPs; see the comparisons of RIACO $\Leftrightarrow$ EIACO, RIACO $\Leftrightarrow$ MIACO and RIACO $\Leftrightarrow$ EIIACO in Table 13.3. This is because both EIACO, MIACO and EIIACO use knowledge, either individual- or environmental-based information, from previous environments to generate immigrant ants, and thus, when not enough time to converge to a good solution is available, it is difficult to transfer useful knowledge, except if the magnitude of change is small, i.e., $m = 0.10$. RIACO generates diversity randomly that is more useful on dynamic cases with $m = 0.50$ and $m = 0.75$, where the changing environments are not similar.

Second, EIACO outperforms RIACO in almost all dynamic cases with $f = 50$ and $m = 0.10$, and $m = 0.25$ in both random and cyclic DTSPs. This is because transferring knowledge makes more sense when the environments are similar. However, if too much knowledge is transferred from the previous environments may lead the population to start from a local optimum solution and get stuck to it, as in the

Table 13.2 Experimental results of algorithms regarding the offline performance for cylic DTSPs

Alg. & Inst.	pr76							
	$f = 5$				$f = 50$			
$m \Rightarrow$	0.10	0.25	0.50	0.75	0.10	0.25	0.50	0.75
RIACO	54082.16	58731.59	69896.53	103643.91	51862.08	54594.27	63953.72	92123.94
EIACO	53560.05	58794.42	70804.71	105641.64	52171.97	54801.34	64591.92	92765.91
HIACO	53601.65	58631.55	70239.99	104357.97	51696.62	54252.89	63789.63	91961.38
MIACO	53698.48	58781.13	70505.11	104851.77	52203.14	54819.25	64559.57	92717.01
EIIACO	53901.79	59191.25	70597.36	105385.63	52162.16	55046.11	64417.57	92590.94
Alg. & Inst.	pr152							
	$f = 5$				$f = 50$			
$m \Rightarrow$	0.10	0.25	0.50	0.75	0.10	0.25	0.50	0.75
RIACO	43123.17	50036.28	59777.25	68348.05	40685.18	45311.61	53565.86	60826.87
EIACO	42440.88	50000.39	59599.67	68250.80	40715.22	45520.32	52889.69	60479.81
HIACO	42714.18	50100.98	59754.37	68432.10	40571.04	45090.22	52844.89	60292.85
MIACO	42397.92	50130.10	59610.09	68134.46	40759.33	45545.47	53002.74	60343.23
EIIACO	42774.22	50365.58	59941.62	68406.41	40925.42	45778.35	53769.09	60479.81
Alg. & Inst.	pr264							
	$f = 5$				$f = 50$			
$m \Rightarrow$	0.10	0.25	0.50	0.75	0.10	0.25	0.50	0.75
RIACO	27118.23	31176.89	38557.06	52909.78	25413.82	28437.01	34043.66	46071.37
EIACO	26463.93	31166.71	38589.28	53356.77	25351.78	28485.02	33875.45	45133.16
HIACO	26850.52	31287.26	38750.83	53190.87	25314.59	28293.39	33877.79	45433.54
MIACO	26510.63	31180.66	38447.30	52951.17	25357.61	28442.63	33914.93	45122.77
EIIACO	26750.38	31192.29	38595.38	53278.29	25556.66	28620.66	34255.57	45930.70

case of pr152 when $f = 50$ and $m = 0.10$ where the RIACO is significantly better than EIACO; see the comparison of RIACO $\Leftrightarrow$ EIACO in Table 13.3. Moreover, RIACO outperforms EIACO in almost all cases of the smallest problem instance, i.e., pr76. This behaviour may have several reasons: (1) the elitism mechanism used in EIACO may not be effective since the environmental changes in a smaller search space have a higher probability to affect the solution that is used to generate guided immigrants for the new environment; (2) random immigrants have a higher probability to hit the optimum in a smaller search space and the risk of randomization is limited, whereas on larger search space it is dangerous; and (3) too much knowledge transferred from previous environments.

Third, HIACO outperforms both RIACO and EIACO in almost all dynamic cases with $f = 50$ in both random and cyclic DTSPs; see comparisons of RIACO $\Leftrightarrow$ HIACO and EIACO $\Leftrightarrow$ HIACO in Table 13.3. This behaviour shows that HIACO inherited the merit of EIACO which is beneficial on slowly changing environments. It can be also observed that the HIACO is significantly better than RIACO even on the smallest problem instance in which EIACO is outperformed. This behaviour shows that HIACO inherited the merit of RIACO. However, HIACO is significantly better than EIACO because it may possibly achieve a good balance between the

Table 13.3 Statistical test results regarding the offline performance of the algorithms for random and cyclic DTSPs

Alg. & Inst.	pr76				pr152				pr264			
Random DTSPs												
$f = 5, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	∼	+	+	+	−	∼	+	+	−	−	+	+
RIACO⇔HIACO	−	+	+	+	−	+	+	+	∼	+	+	+
RIACO⇔MIACO	+	+	+	+	∼	+	+	+	−	+	+	+
RIACO⇔EIIACO	+	+	+	+	+	+	+	+	−	∼	+	+
EIACO⇔HIACO	−	−	−	−	∼	∼	−	−	+	+	∼	−
EIACO⇔MIACO	+	+	+	+	+	+	+	+	+	+	+	+
EIACO⇔EIIACO	+	+	∼	−	+	+	∼	∼	+	+	−	−
HIACO⇔MIACO	+	+	+	+	+	+	+	+	−	∼	+	+
HIACO⇔EIIACO	+	+	+	+	+	+	+	+	−	−	−	+
MIACO⇔EIIACO	+	−	−	−	+	∼	−	−	+	−	−	−
$f = 50, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	+	+	+	+	+	−	−	−	−	−	−	−
RIACO⇔HIACO	−	−	−	∼	∼	−	−	−	−	−	−	−
RIACO⇔MIACO	+	+	+	+	+	+	−	−	∼	∼	−	−
RIACO⇔EIIACO	+	+	+	+	+	+	∼	∼	+	+	+	∼
EIACO⇔HIACO	−	−	−	−	−	−	∼	∼	−	−	∼	+
EIACO⇔MIACO	∼	∼	+	+	∼	+	+	+	+	+	+	+
EIACO⇔EIIACO	∼	−	∼	−	∼	+	+	+	+	+	+	+
HIACO⇔MIACO	+	+	+	+	+	+	+	+	+	+	+	+
HIACO⇔EIIACO	+	+	+	+	+	+	+	+	+	+	+	+
MIACO⇔EIIACO	∼	−	∼	−	∼	+	+	+	+	+	+	+
Cyclic DTSPs												
$f = 5, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	−	∼	+	+	−	∼	−	−	−	∼	+	+
RIACO⇔HIACO	−	−	+	+	−	+	∼	+	−	+	+	+
RIACO⇔MIACO	−	∼	+	+	−	+	−	−	−	∼	−	∼
RIACO⇔EIIACO	−	+	+	+	−	+	+	∼	−	∼	+	+
EIACO⇔HIACO	∼	−	−	−	+	+	+	+	+	+	+	−
EIACO⇔MIACO	+	∼	−	−	∼	+	∼	−	∼	∼	−	−
EIACO⇔EIIACO	+	+	−	−	+	+	+	+	+	+	∼	−
HIACO⇔MIACO	+	+	+	+	−	∼	−	−	−	−	−	−
HIACO⇔EIIACO	+	+	+	+	+	+	+	∼	−	−	−	+
MIACO⇔EIIACO	+	+	+	+	+	+	+	+	+	∼	+	+
$f = 50, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	+	+	+	+	∼	+	−	−	−	+	−	−
RIACO⇔HIACO	−	−	−	−	−	−	−	−	−	−	−	−
RIACO⇔MIACO	+	+	+	+	+	+	−	−	−	∼	−	−
RIACO⇔EIIACO	+	+	+	+	+	+	+	+	+	+	+	−
EIACO⇔HIACO	−	−	−	−	−	−	∼	−	∼	−	∼	+
EIACO⇔MIACO	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
EIACO⇔EIIACO	∼	+	−	−	+	+	+	+	+	+	+	+
HIACO⇔MIACO	+	+	+	+	+	+	+	∼	∼	+	+	−
HIACO⇔EIIACO	+	+	+	+	+	+	+	+	+	+	+	+
MIACO⇔EIIACO	∼	+	∼	∼	+	+	+	+	+	+	+	+

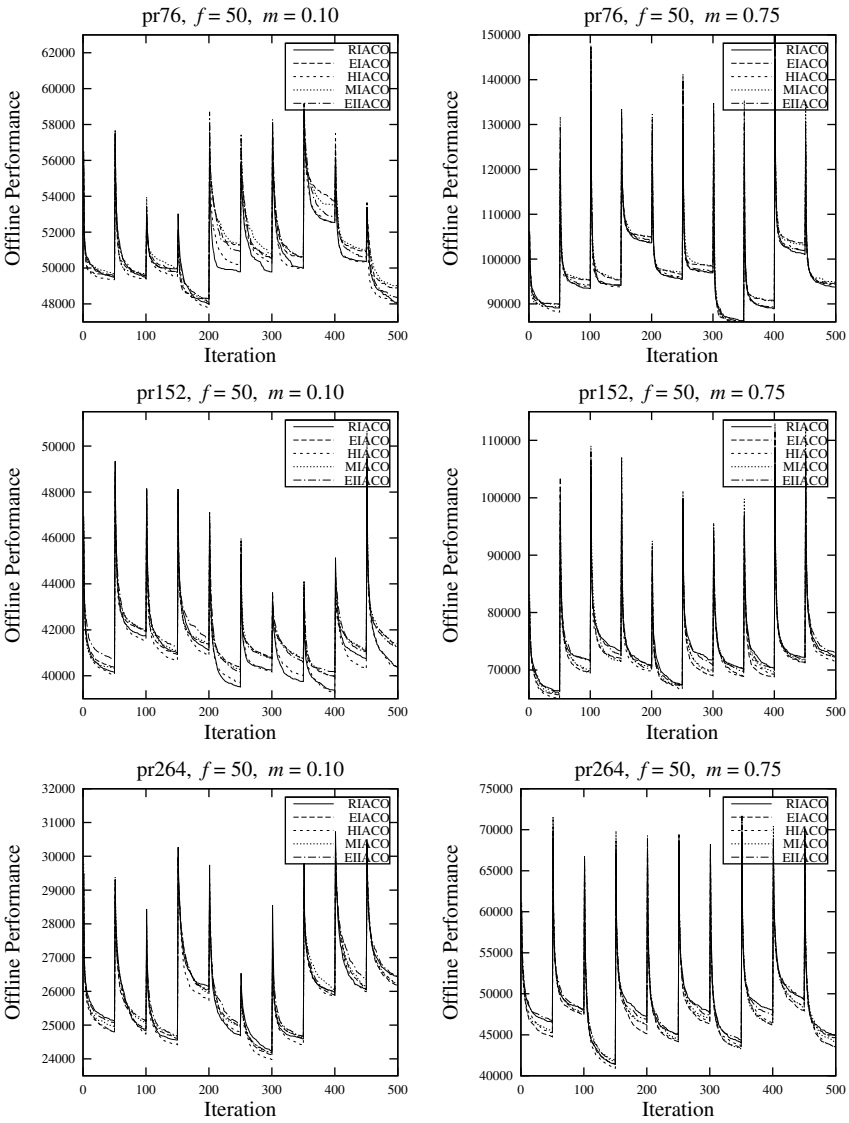

Fig. 13.3 Dynamic behaviour of the investigated ACO algorithms on random DTSPs

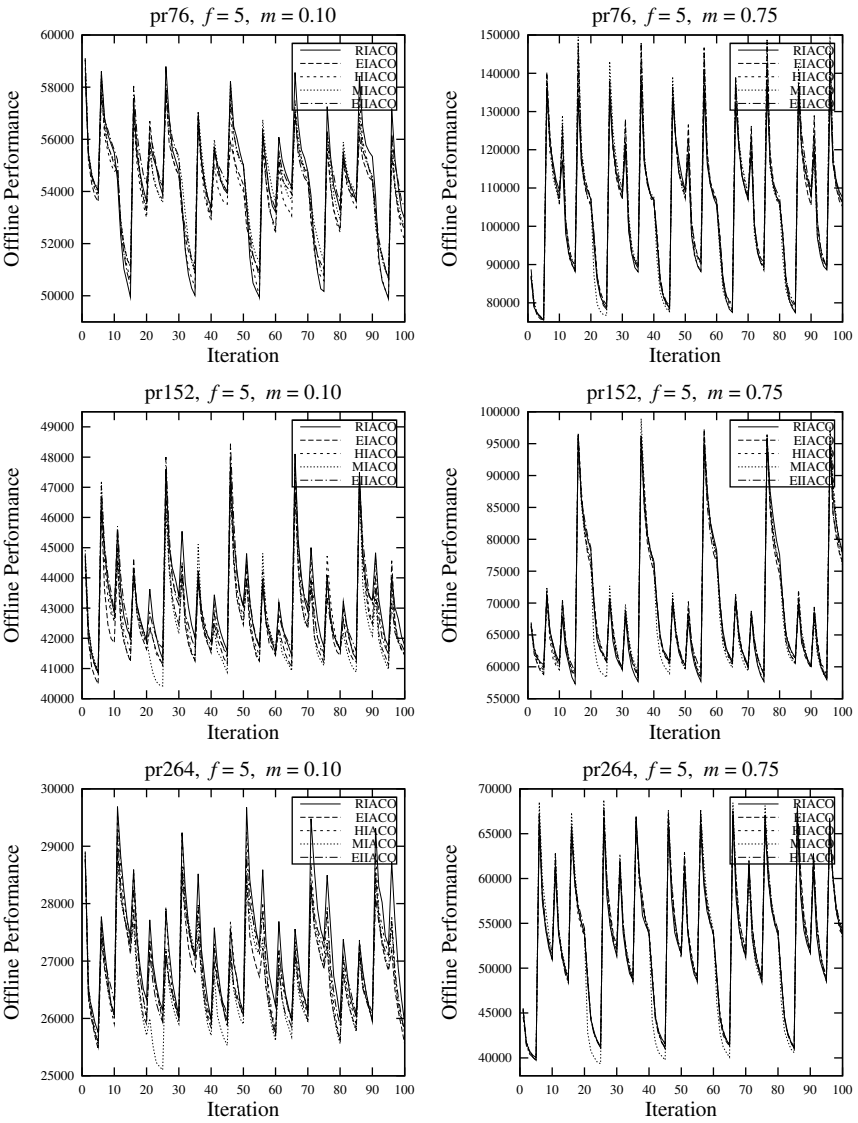

Fig. 13.4 Dynamic behaviour of the investigated ACO algorithms on cyclic DTSPs

diversity level and the knowledge transferred. Moreover, in cases with $f = 5$, HIACO outperforms EIACO, whereas it has a similar behaviour with RIACO, but slightly degraded. On the other hand, HIACO outperforms EIIACO in all dynamic cases except on the largest problem instance, i.e., `pr264`, with $m = 0.10$, 0.25 and 0.5; see the comparisons of HIACO $\Leftrightarrow$ EIIACO in Table 13.3. A similar observation of this behaviour has been found in [32], where the performance of

EIIACO was promoted as the size of the problem instance increased. This because of the diversity level generated from the environmental-information based immigrants, which gather more information than the individual-information based immigrants, i.e., in EIACO and MIACO.

Fourth, MIACO has a similar behaviour with EIACO when compared with other algorithms, in random DTSPs. However, it is outperformed by EIACO and HIACO in almost all dynamic cases; see the results regarding MIACO $\Leftrightarrow$ EIACO and HIACO $\Leftrightarrow$ MIACO in Table 13.3 and the results in Table 13.1. However, in cyclic DTSPs, MIACO outperforms EIACO and HIACO in cases where $f = 5$ and $m = 0.50$ and 0.75, whereas it is underperformed in cases where $f = 5$ and $m = 0.10$ and 0.25; see the results regarding MIACO $\Leftrightarrow$ EIACO and HIACO $\Leftrightarrow$ MIACO in Table 13.3 and the results in Table 13.2. Furthermore, MIACO and EIACO are insignificant different in cyclic DTSPs where $f = 50$, while HIACO significantly outperforms them. This is because EIACO is beneficial when the changing environment is similar, either in cyclic or random DTSPs, and MIACO is beneficial when the changing environment re-appears, i.e., cyclic DTSPs. The reason why EIACO is effective in slightly changing environments is explained above. The reason why MIACO is effective in cyclic DTSPs is that it can move the population directly to a previously visited environment. MIACO stores the best solutions for all cyclic base states and reuses them by generating memory-based immigrants. Moreover, on the smallest problem instance, i.e., `pr76`, RIACO outperforms MIACO in almost all dynamic cases, either random or cyclic DTSPs. This is because of the same reasons discussed on EIACO above, since both algorithms use the elitist mechanism to generate immigrants.

Finally, EIIACO outperforms RIACO in almost all dynamic cases with $f = 5$ and $m = 0.10$, while it is underperformed in dynamic cases with $f = 50$; see the comparisons of RIACO $\Leftrightarrow$ EIIACO in Table 13.3 for both random and cyclic DTSPs. This is because EIIACO transfers knowledge from previous environment and it is beneficial when the changing environments are similar. On the other hand, if too much knowledge is transferred the performance is degraded as in EIACO. Moreover, EIIACO outperforms EIACO and MIACO in $f = 5$ and $m = 0.50$ and 0.75, while it is underperformed in dynamic cases with $f = 50$. However, on the smallest problem instance, i.e., `pr76`, EIIACO overcomes the issue of EIACO and MIACO, which they have a degraded performance, since it is significantly better. However, in cyclic DTSPs, MIACO outperforms EIIACO in almost all dynamic cases, as expected; see the comparisons of EIACO $\Leftrightarrow$ EIIACO and MIACO $\Leftrightarrow$ EIIACO in Table 13.3 for both random and cyclic DTSPs.

13.5.4 Experimental Results and Analysis of the Investigated Algorithms with Other Peer ACO

In this section, we compare the offline performance of the investigated algorithms above, with several other existing peer ACO proposed in the literature for different DTSPs. These peer ACO algorithms are S-ACO and P-ACO described in Section

Table 13.4 Experimental results regarding offline performance on the DTSP with $m = rand[0,1]$ and $f = rand[1,100]$ in `lin318`

Algorithms	S-ACO	P-ACO	M-ACO	RIACO	EIACO	HIACO	MIACO	EIIACO
Offline Performance								
Mean	35008.25	34960.87	34845.93	34200.89	33794.34	33875.42	33904.18	34215.37
(Std Dev)	96.07	186.63	141.75	140.23	155.60	148.64	163.62	169.32
t-test Results								
S-ACO		$\sim$	$+$	$+$	$+$	$+$	$+$	$+$
P-ACO	$\sim$		$+$	$+$	$+$	$+$	$+$	$+$
M-ACO	$-$	$-$		$+$	$+$	$+$	$+$	$+$
RIACO	$-$	$-$	$-$		$+$	$+$	$+$	$\sim$
EIACO	$-$	$-$	$-$	$-$		$-$	$-$	$-$
HIACO	$-$	$-$	$-$	$-$	$+$		$\sim$	$-$
MIACO	$-$	$-$	$-$	$-$	$+$	$\sim$		$-$
EIIACO	$-$	$-$	$-$	$\sim$	$+$	$+$	$+$	

13.3, and Memetic ACO (M-ACO) [30], which is a hybridization of P-ACO and an adaptive inversion local search.

In the previous experiments, we have investigated the offline performance and dynamic behaviour of ACO algorithms with immigrants schemes under two kinds of dynamic environments, i.e., random and cyclic DTSPs, but with fixed values of f and m. However, in real-world problems both f and m may vary during the execution of the algorithm. In order to investigate the behaviour of the investigated algorithms and compare them with existing algorithms in such kinds of environments, further experiments were carried out in `lin318`. The values f and m were generated randomly with a uniform distribution in $[1,100]$ and $[0,1]$, respectively. Since the time interval of such kind of environment varies, many existing approaches used in ACO algorithms for DTSPs, e.g. global and local restart strategies and diversity maintenance schemes [16, 21], cannot be applied, since they do not have any mechanism to detect dynamic changes.

The experimental settings and performance measure were the same as in previous experiments. The experimental results regarding the offline performance are presented in Table 13.4 with the corresponding two-tailed *t*-test results with 58 degrees of freedom at a 0.05 level of significance, "$+$" or "$-$" indicates that the algorithm in the column is significantly better or the algorithm in the row is significantly better, respectively, where $\sim$ indicates no significance between the algorithms. Moreover, the values of varying f and m are plotted in Fig. 13.5 and the corresponding dynamic behaviour of the algorithms is plotted in Fig. 13.6. From the experimental results, several observations can be drawn.

First, the results in Table 13.4 almost match the analysis of our previous experiments, where EIACO and MIACO are significantly better than RIACO, whereas EIIACO is significantly worst than RIACO. HIACO improves the performance of RIACO, whereas it is worst than EIACO. EIACO is the champion algorithm from all the ACO algorithms with immigrants schemes. This is because the generated environment is a DTSP with random traffic factors, where EIACO perform well.

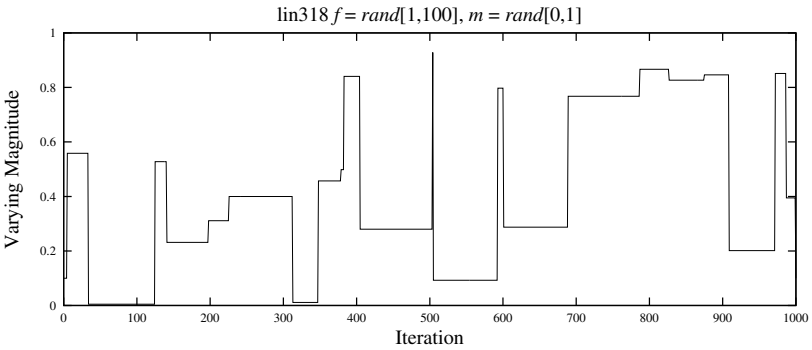

Fig. 13.5 Varying values for $m = rand[0,1]$ and $f = rand[1,100]$ used for the DTSP

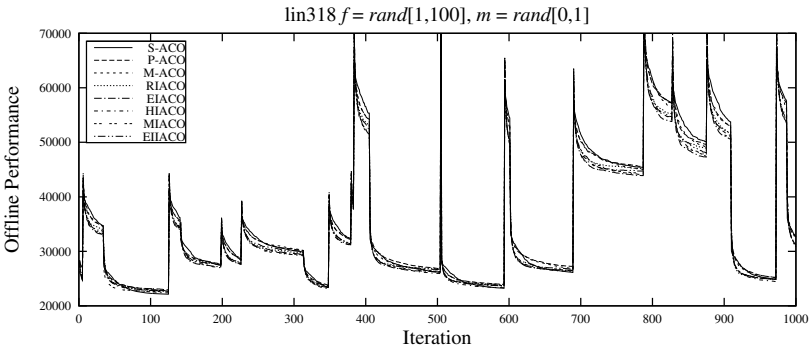

Fig. 13.6 Dynamic behaviour of the investigated ACO algorithms in comparison with other peer ACO algorithms on the DTSP with $m = rand[0,1]$ and $f = rand[1,100]$

MIACO is not performing well since it is not guaranteed that previously visited environments will reappear again.

Second, all investigated algorithms outperform other peer ACO algorithms. This is because the S-ACO uses only pheromone evaporation to eliminate pheromone trails from the previous environment that are not useful to the new one, and thus, needs sufficient time to adapt to the changing environments. On the other hand, P-ACO eliminates pheromone trails directly if an ant is removed from k_{long}. However, if identical ants are stored in the k_{long}, then the algorithm will reach stagnation behaviour, and thus, needs sufficient time to escape from it. M-ACO is significantly better than both S-ACO and P-ACO since the local search that is integrated with ACO promotes exploitation to improve the solution quality, and the risk of stagnation is eliminated using a diversity maintenance scheme based on traditional immigrants. Whenever, k_{long} reaches stagnation behaviour a random immigrant replaces an ant until the algorithm generates sufficient diversity.

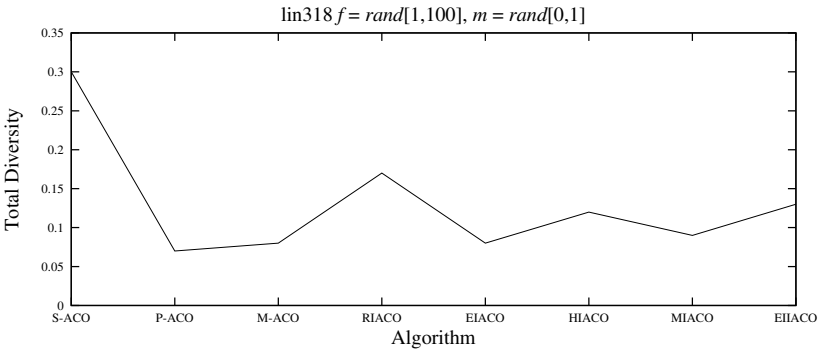

Fig. 13.7 Total diversity of the investigated ACO algorithms in comparison with other peer ACO algorithms on the DTSP with $m = rand[0,1]$ and $f = rand[1,100]$

Finally, in order to investigate the effect of immigrants scheme on the population diversity, we calculate the mean population diversity of all iterations as follows:

$$Div = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N}\left(\frac{1}{\mu(\mu-1)}\sum_{p=1}^{\mu}\sum_{q\neq p}^{\mu}M_{ij}\right)\right), \tag{13.10}$$

where G is the number of iterations, N is the number of runs, μ is the size of the population and M_{ij} is the common edges between ant p and ant q as defined in Eq. (13.7). A value closer to 0 means that the ants are identical and a value closer to 1 means that the ants are completely different. The total diversity results for the different dynamic cases are presented in Fig. 13.7. It can be observed that S-ACO has a higher diversity than all algorithms. The P-ACO algorithm has the lowest diversity level which shows the effect when identical ants are stored in the population-list. RIACO maintains the highest diversity among the remaining algorithms with different immigrants schemes, since diversity is generated randomly, whereas the remaining algorithms generate guided diversity via transferring knowledge. However, EIIACO maintains higher diversity than both EIACO and MIACO which shows that the environmental-information immigrants generate higher diversity than individual-information immigrants. HIACO has a lower diversity than RIACO but higher than EIACO which shows that it has inherited the merits of the two immigrants schemes. Considering the results of the total diversity with the those of the offline performance shows that ACO algorithms that maintain high diversity levels do not always achieve better performance than other ACO algorithms for the DTSP; see Table 13.4 and Fig. 13.6.

13.6 Conclusions and Future Work

Several immigrants schemes have been successfully applied to ACO algorithms to address different DTSPs [29, 31, 32]. Immigrant ants are generated to transfer

knowledge to the pheromone trails and maintain diversity. In this chapter, we re-investigate those algorithms, on the way they generate immigrant ants, and apply them to DTSP with traffic factors. We generate two types of dynamic environments: (1) the traffic factor changes randomly; and (2) the traffic factor changes in a cyclic pattern where the environments will reappear.

From the experimental results of comparing the investigated algorithms on different cases of DTSPs and with other peer ACO algorithms, the following concluding remarks can be drawn. First, immigrants schemes enhance the performance of ACO for DTSPs. Second, RIACO is advantageous in fast and significantly changing environments. Third, EIACO is advantageous in slowly and slightly changing environments. Fourth, HIACO promotes the performance of EIACO in slowly changing environments, while it slightly degrades the performance of RIACO in fast changing environments. Fourth, MIACO is advantageous in cyclic changing environments, where previous environments will re-appear. Fifth, EIIACO promotes the performance of EIACO in environments with significant changes. Sixth, transferring too much knowledge from previous environments may degrade the performance. Finally, a high level of diversity do not always enhance the performance of ACO in DTSPs.

In general, almost all ACO algorithms based on immigrants schemes outperform other peer ACO algorithms. Furthermore, different immigrants schemes are beneficial for different dynamic environmental cases.

For future work, it will be interesting to hybridize more types of immigrants schemes, to achieve a good balance between the knowledge transferred and the diversity maintenance. Another future work is to apply the proposed algorithms to other relevant problems, e.g., the dynamic vehicle routing problem, and in more challenging environments, where, apart from dynamic changes, a small amount of noise may be generated every iteration.

Acknowledgements. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of U.K. under Grant numbers EP/E060722/1, EP/E060722/2, and EP/K001310/1.

References

[1] Angus, D.: Niching for population-based ant colony optimization. In: Proc. of the 2nd IEEE Inter. Conf. on e-Science and Grid Comp., pp. 15–22 (2006)
[2] Bell, J.E., McMullen, P.R.: Ant colony optimization techniques for the vehicle routing problem. Advanced Engineering Informatics 18, 41-48 (2004)
[3] Bullnheimer, B., Haïti, R., Strauss, C.: An improved ant system algorithm for the vehicle routing problem. Ann. Oper. Res. **89**(1), 319-328 (1999)
[4] Bullnheimer, B., Hartl, R.F., Strauss, C.: A new rank based version of the ant system - a computational study. Central Eur. J. for Oper. Res. in Economics **7**(1), 25–38 (1999)
[5] Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York (1999)
[6] Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: Proc. 1999 IEEE Congr. Evol. Comput. vol. 3, pp. 1875–1882 (1999)

[7] Cruz, C., González, J.R., Pelta, D.A.: Optimization in dynamic environments: a survey on problems, methods and measures. Soft Comput., **15**(7), 1427–1448 (2011)
[8] Cobb, H.G., Grefenstette, J.J.: Genetic algorithms for tracking changing environments. In: Proc. 5th Int. Conf. on Genetic Algorithm, pp. 523–530 (1993)
[9] Colorni, A., Dorigo, M., Maniezzo, V.: Distributed optimization by ant colonies. In: Proc. 1st Europ. Conf. on Artif. Life, pp. 134–142 (1992)
[10] Cordón, O., de Viana, I.F., Herrera, F., Moreno, L.: A new ACO model integrating evolutionary computation concepts: The best worst Ant System. In: Proc. 2nd Int. Workshop on Ant Algorithms, pp. 22-29 (2000)
[11] Di Caro, G., Dorigo, M.: Ant Net: Distributed Stigmergetic Control for Communications Networks. J. of Artif. Intell. Res. **9**(1), 317–365 (1998)
[12] Di Caro, G., Ducatelle, F., Gambardella, L.M.: AntHocNet: An ant-based hybrid routing algorithm for mobile ad hoc networks. In: Proc. 8th Int. Conf. on Parallel Problem Solving from Nature, LNCS 3242, pp. 461–470 (2004)
[13] Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst., Man and Cybern., Part B: Cybern. **26**(1), 29–41 (1996)
[14] Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evol. Comput. **1**(1), 53–66 (1997)
[15] Dorigo, M., Stützle, T.: Ant Colony Optimization. The MIT Press, London (2004)
[16] Eyckelhof, C.J., Snoek, M.: Ant Systems for a Dynamic TSP. In: Proc. 3rd Int. Workshop on Ant Algorithms, pp. 88–99 (2002)
[17] Gambardella, L.M., Taillard, E.D., Dorigo, M.: Ant colonies for the quadratic assignment problem. J. of the Oper Res Society **50**, 167–176 (1999)
[18] Grefenestette, J.J.: Genetic algorithms for changing environments. In: Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, pp. 137–144 (1992)
[19] Guntsch, M., Middendorf, M.: Applying population based ACO to dynamic optimization problems. In: Proc. 3rd Int. Workshop on Ant Algorithms, LNCS 2463, pp. 111–122 (2002)
[20] Guntsch, M., Middendorf, M.: A population based approach for ACO. In: EvoWorkshops 2002: Appl. of Evol. Comput., pp. 72–81 (2002)
[21] Guntsch, M., Middendorf, M.: Pheromone modification strategies for ant algorithms applied to dynamic TSP. In: EvoWorkshops 2001: Appl. of Evol. Comput., pp. 213-222 (2001)
[22] Guntsch, M., Middendorf, M., Schmeck, H.: An ant colony optimization approach to dynamic TSP. In: Proc. 2001 Genetic and Evol. Comput. Conf., pp. 860–867 (2001)
[23] Guo, T., Michalewicz, Z.: Inver-over operator for the TSP. In: Proc. 5th Int. Conf. on Parallel Problem Solving from Nature, LNCS 1498, pp. 803–812 (1998)
[24] He, J., Yao, X.: From an individual to a population: An analysis of the first hitting time of population-based evolutionary algorithms. IEEE Trans. Evol. Comput. **6**(5), 495–511 (2002)
[25] Holland, J.: Adaption in Natural and Artificial Systems, University of Michigan Press (1975)
[26] Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments - a survey. IEEE Trans. Evol. Comput. **9**(3), 303–317 (2005)
[27] Lin, S., Kerninghan, B.W.: An effective heuristic algorithm for the traveling salesman problem. Oper. Res. **21**(2), 498–516 (1973)
[28] Maniezzo, V., Colorni, A.: The ant system applied to the quadratic assignment problem. IEEE Trans. Knowledge and Data Engineering **9**(5), 769–778 (1999)

[29] Mavrovouniotis, M., Yang, S.: Ant colony optimization with immigrants schemes for dynamic environments. In: Proc. 11th Int. Conf. on Parallel Problem Solving from Nature, LNCS 6239, pp. 371–380 (2010)
[30] Mavrovouniotis, M., Yang, S.: A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. Soft Comput. **15**(7), 1405–1425 (2011)
[31] Mavrovouniotis, M., Yang, S.: Memory-based immigrants for ant colony optimization in changing environments. In: EvoWorkshops 2011: Appl. of Evol. Comput., LNCS 6624, pp. 324–333 (2011)
[32] Mavrovouniotis, M., Yang, S.: An immigrants scheme based on environmental information for ant colony optimization for the dynamic travelling salesman problem. In: Proc. 10th Int. Conf. Evolution Artificial, LNCS 7401, pp. 1-12 (2011)
[33] Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, Berlin, third edition (1999)
[34] Montemanni, R., Gambardella, L., Rizzoli, A., Donati, A.: An new algorithm for a dynamic vehicle routing problem based on ant colony system. In: Proc. 2nd Int. Workshop on Freight Transportation and Logistics, pp. 27–30 (2003)
[35] Neumann, F., Witt, C.: Runtime analysis of a simple ant colony optimization algorithm. Algorithmica **54**(2), 243–255 (2009)
[36] Rizzoli, A. E., Montemanni, R., Lucibello, E., Gambardella, L. M.: Ant colony optimization for real-world vehicle routing problems - from theory to applications. Swarm Intell. **1**(2), 135–151 (2007)
[37] Stützle, T., Hoos, H.: The MAX-MIN ant system and local search for the traveling salesman problem. In: Proc. 1997 IEEE Int. Conf. Evol. Comput., pp. 309–314 (1997)
[38] Stützle, T., Hoos, H.: MAX-MIN Ant System. Future Generation Computer Systems **8**(16), 889–914 (2000)
[39] Yang, S.: Memory-based immigrants for genetic algorithms in dynamic environments. In: Proc. 2005 Genetic and Evol. Conf., vol. 2, pp. 1115–1122 (2005)
[40] Yang, S.: Genetic algorithms with elitism based immigrants for changing optimization problems. In: EvoWorkshops 2007: Appl. of Evol. Comput., LNCS 4448, pp. 627–636 (2007)
[41] Yang, S.: Genetic algorithms with memory and elitism based immigrants in dynamic environments. Evol. Comput. **16**(3), 385–416 (2008)
[42] Yang, S., Cheng, H., Wang, F.: Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile ad hoc networks. IEEE Trans. Syst., Man, and Cybern. Part C: Appl. and Rev. **40**(1), 52-63 (2010)
[43] Yang, S., Tinos, R.: A hybrid immigrants scheme for genetic algorithms in dynamic environments. Int. J. of Autom. and Comput. **4**(3), 243–254 (2007)
[44] Yang, S., Yao, X.: Population-based incremental learning with associative memory for dynamic environments. IEEE Trans. Evol. Comput. **12**(5), 542–561 (2008)
[45] Yu, X., Tang, K., Yao, X.: An immigrants scheme based on environmental information for genetic algorithms in changing environments. In: Proc. 2008 IEEE Congr. Evol. Comput., pp. 1141–1147 (2008)
[46] Yu, X., Tang, K., Yao, X.: Immigrant schemes for evolutionary algorithms in dynamic environments: Adapting the replacement rate. Sci. China Series F: Inf. Sci. **53**(1), 1–11 (2010)
[47] Yu, X., Tang, K., Chen, T., Yao, X.: Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization. Memetic Comput. **1**(1), 3–24 (2009)