Applied Soft Computing 13 (2013) 4023–4037

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors

Michalis Mavrovouniotis*, Shengxiang Yang

Centre for Computational Intelligence (CCI), School of Computer Science and Informatics, De Montfort University, The Gateway, Leicester LE1 9BH, United Kingdom

ARTICLE INFO

Article history:
Received 1 December 2012
Received in revised form 23 April 2013
Accepted 28 May 2013
Available online 10 June 2013

Keywords:
Ant colony optimization
Immigrants schemes
Dynamic optimization problem
Dynamic travelling salesman problem
Traffic factor

ABSTRACT

Traditional ant colony optimization (ACO) algorithms have difficulty in addressing dynamic optimization problems (DOPs). This is because once the algorithm converges to a solution and a dynamic change occurs, it is difficult for the population to adapt to a 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 to address this problem is to increase the diversity via transferring knowledge from previous environments to the pheromone trails using immigrants schemes. In this paper, an ACO framework for dynamic environments is proposed where different immigrants schemes, including random immigrants, elitism-based immigrants, and memory-based immigrants, are integrated into ACO algorithms for solving DOPs. From this framework, three ACO algorithms, where immigrant ants are generated using the aforementioned immigrants schemes and replace existing ants in the current population, are proposed and investigated. Moreover, two novel types of dynamic travelling salesman problems (DTSPs) with traffic factors, i.e., under random and cyclic dynamic environments, are proposed for the experimental study. The experimental results based on different DTSP test cases show that each proposed algorithm performs well on different environmental cases and that the proposed algorithms outperform several other peer ACO algorithms.

1. Introduction

Ant colony optimization (ACO) algorithms emulate the behaviour of real ant colonies when they search for food from their nest to food sources. Ants communicate via their pheromone trails in order to complete this “food-searching” task as efficiently as possible. ACO algorithms have proved to be effective optimization methods in many real-world applications [16,17,20,42].

Traditionally, researchers have focused on ACO to address static optimization problems (SOPs), e.g., the travelling salesman problem (TSP). For SOPs, the environment remains fixed during the execution of algorithms [3,5,34]. However, in many real-world problems, we have to deal with dynamic environments [31]. For dynamic optimization problems (DOPs), the problem may change over time regarding the objective function, decision variables, problem instance, constraints, and so on. Such uncertainties may change/move the optimum, and, thus, the problem becomes more challenging and has much in common with many real-world scenarios. The aim, when solving SOPs, is to obtain an optimal or near-optimal solution efficiently, whereas for DOPs the aim is to track the optimal solution over time with the environmental changes.

ACO algorithms can adapt to dynamic changes since they are inspired from nature, which is a continuous adaptation process [31]. More precisely, they can adapt to dynamic changes by transferring knowledge from past environments [7]. However, traditional ACO algorithms cannot adapt well to dynamic changes once the ants reach the stagnation behaviour, where ants follow the same path, since the pheromone will be distributed into a single trail. Therefore, using the pheromone trails of the previous environment in the new environment will not be sufficient since ants will be biased by the previous path even after a dynamic change. The algorithm loses its adaptation capability since it does not maintain the diversity within the population. A simple way to address this problem is to consider every change as the arrival of a new problem instance which needs to be solved from scratch and re-initialize the pheromone trails. However, this restart strategy is computationally expensive and usually not efficient.

Recently, developing strategies for ACO algorithms to enhance their performance for DOPs and increase their adaptation capabilities has attracted a lot of attention [11]. The strategies developed

* Corresponding author. Tel.: +44 (0) 116 250 6757.
E-mail addresses: mmavrovouniotis@dmu.ac.uk (M. Mavrovouniotis), syang@dmu.ac.uk (S. Yang).

1568-4946/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2013.05.022

include: (1) local and global restart strategies [27]; (2) pheromone manipulation schemes to maintain or increase diversity [21,35,37]; (3) memory-based approaches [25,28]; and (4) memetic algorithms [36]. These methods have been applied to different types of dynamic TSPs (DTSPs) due to its importance for many real-world applications [39,42].

As we have seen in many evolutionary algorithms (EAs) for binary-encoded DOPs, immigrants schemes are effective when applied to different DOPs [49,51,55,60]. Immigrants enable an EA to maintain the diversity of the population at a certain level, by introducing new individuals into the current population [24]. A good start has been made on ACO algorithms for permutation-encoded DOPs, where immigrant ants are generated to the population [35,37]. In [35], the random immigrants ACO (RIACO) and elitism-based immigrants ACO (EIACO) have been applied on a DTSP where the cities are exchanged with other cities stored in a spare pool over time. In [37], the memory-based immigrants ACO (MIACO) has been applied on a DTSP with traffic factors. The difference between RIACO, EIACO, and MIACO lies on the way immigrants are generated. In RIACO, EIACO, and MIACO, immigrants are generated randomly, using the best ant from the previous environment as the base to generate immigrants, and using the best ant from a memory as the base to generate immigrants, respectively.

In this paper, we further investigate the performance of ACO algorithms with immigrants schemes on different dynamic test cases. RIACO, EIACO and MIACO are applied on two novel variations of DTSPs, e.g., DTSP with random and cyclic traffic factors, and they are systematically constructed from several static TSP problem instances. A series of different DTSPs are generated to investigate the weaknesses and strengths of RIACO, EIACO, and MIACO for solving DTSPs. This paper also carries out experiments on sensitivity analysis with respect to several important parameters, such as the immigrants replacement rate, the size of the memory structures used in the proposed framework and the range of the traffic factors, on the performance of the investigated ACO algorithms for DTSPs. Moreover, the proposed algorithms are compared against several peer ACO algorithms on DTSPs.

The rest of the paper is organized as follows. Section 2 describes the field of dynamic optimization. Section 3 describes the proposed DTSPs variations. Section 4 describes a conventional ACO for DOPs. Section 5 describes existing approaches on different DTSPs. Section 6 first describes the framework of ACO algorithms with immigrants schemes and then describes the proposed algorithms. Section 7 presents and analyzes several experimental studies for different key parameters and compares the proposed algorithms with existing ones. Finally, Section 8 concludes this paper with discussions on the directions for future work.

2. Dynamic optimization

2.1. A brief introduction

A DOP can be intuitively defined as a sequence of several static problem instances that are linked under some dynamic rules. The main aspects of "dynamism" are the frequency and magnitude of environmental changes. The former corresponds to the speed at which environmental changes occur and the latter corresponds to the degree of environmental changes. An environmental change may involve factors like the objective function, input variables, problem instance, constraints, and so on.

Formally, a DOP can be defined as follows:

$$\Pi = (X(t), \Omega(t), f(t))_{t \in T}, \tag{1}$$

where Π is the optimization problem, $X(t)$ is the search space, $\Omega(t)$ is a set of constraints, $f(t)$ is the objective function, which assigns an objective value to each solution $x \in X(t)$, where all of them are assigned with a time value t, and T is a set of time values.

2.2. Approaches to dynamic optimization

Mainly, the research to address DOPs is focused on EAs, which is otherwise known as evolutionary dynamic optimization (EDO). EAs are able to transfer knowledge since the information of previous environment is stored in the population of individuals of the previous iterations. However, the individuals in the old environment may not be feasible for the new one. But, if they are re-evaluated or repaired, they may transfer valuable information. Several surveys are available for EDO [31,41,54].

For DOPs, once the population converges to an optimum, then it is difficult for the population to track the moving optimum after a change. Many strategies have been proposed and integrated with EAs to improve the re-optimization time and maintain a high quality of the output efficiently, simultaneously. The main contributions of these strategies can be categorized as: (1) increasing diversity after a dynamic change [12,46]; (2) maintain diversity during the execution via immigrants schemes [24,49,51,55,60]; (3) memory-based schemes [50,59]; (4) multi-population approaches [9,45,48]; and (5) hybrid and memetic algorithms [49,47].

2.3. Immigrants schemes for EAs

Among the approaches described above, immigrants schemes have been found beneficial when applied to EAs for many binary-encoded DOPs [49,51,55,60,62]. Immigrants schemes work by replacing a predefined portion of individuals in the current population with new immigrants in each generation. Random immigrants were introduced to EAs in order to increase the diversity within the population and enhance their performance for DOPs [24]. According to the experiments in [13], random immigrants perform well when dynamic changes occur frequently and the changing environments are not similar.

Elitism- and memory-based immigrants were introduced to EAs in order to address DOPs with slightly and slowly changing environments [49,51]. For the elitism-based immigrants scheme, the best individual from the previous population is used as the base to generate immigrants via mutation, while for the memory-based immigrants scheme, the best individual from an external memory is used as the base to generate immigrants via mutation. Moreover, hybrid immigrants were proposed [55], where random and elitism-based immigrants are combined to improve the performance of elitism-based immigrants in significantly changing environments, while they degrade the performance in slightly changing environments.

2.4. Benchmark DOPs

Much research on dynamic optimization has been done with EAs on binary-encoded DOPs, such as the one-max problem, royal road problem, plateau problem, and knapsack problem [49,60]. Usually, the XOR DOP generator proposed in [56] is used to convert any binary-encoded static problem into DOPs with different properties.

On the other hand, the research on ACO for DOPs is focused on permutation-encoded problems, such as TSPs and vehicle routing problems (VRPs). It is fair to say that the research on ACO for permutation-encoded DOPs is still in its infancy when compared with the research on EAs for DOPs. For example, the quantity of contributions on EAs for DOPs is massive, whereas almost all contributions on ACO for DOPs are described in Section 5.

Table 1
Mathematical symbols used in this paper.

Symbol	Description
$G=(V, E)$	Fully connected weighted graph
V	Set of vertices
E	Set of edges
n	Number of cities (vertices)
d_{ij}	Distance (or travel time) between cities i and j
t_{ij}	Traffic factor between cities i and j
(i, j)	Connection between cities i and j
f	Frequency of dynamic change
m	Magnitude of dynamic change
F_L	Minimum limit of the traffic factor
F_U	Maximum limit of the traffic factor
R	Random number
ρ	Pheromone evaporation rate
q_0	Decision rule parameter to adjust exploration/exploitation
t	Current iteration count
$P(t)$	Population of ants for iteration t
τ_0	Initial pheromone trail value
τ_{max}	Maximum pheromone trail value
$k_{long}(t)$	Long-term memory for iteration t
K_l	Size of the long-term memory
$k_{short}(t)$	Short-term memory for iteration t
K_s	Size of the short-term memory
r	Immigrant ant replacement rate
S_{ri}	Set of random immigrant ants
S_{ei}	Set of elitism-based immigrant ants
S_{mi}	Set of memory-based immigrant ants
t_M	Iteration number where the next memory update will occur
μ	Ant's population size
r_C	Random city
x^{bs}	Global best solution
x^{ib}	Iteration best solution
x^{elite}	Best solution for a given environment
x^r	Random ant
x^{cm}	Most similar ant in long-term memory
x^{mb}	Best ant in long-term memory

3. Dynamic travelling salesman problems

3.1. Static TSPs

The TSP is one of the most fundamental $\mathcal{NP}$-complete combinatorial optimization problems (COPs). 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 $V=\{0, \ldots, n\}$ is a set of vertices and $E=\{(i, j): i \neq j\}$ 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 (i, j) is associated with a non-negative value d_{ij} which represents the distance between cities i and j.

Formally, the TSP can be described as follows:

$$f(x) = min\sum_{i=0}^{n}\sum_{j=0}^{n} d_{ij}\psi_{ij}, \tag{2}$$

subject to:

$$\psi_{ij} = \begin{cases} 1, \text{if}(i, j)\text{is used in the tour}, \\ 0, \text{otherwise}, \end{cases} \tag{3}$$

where $\psi_{ij} \in \{0, 1\}$, n is the number of cities, and d_{ij} is the distance between city i and j. For the convenience of readers we list some key mathematical symbols used in the paper in Table 1.

3.2. Dynamic TSPs

Traditionally, researchers have focused on the static TSP, in which the problem remains unchanged during the execution of an algorithm. In evidence, a lot of algorithms, either exact algorithms or approximation algorithms (or heuristics), have been proposed to solve the static TSP [33,38,43].

However, in many real-world applications, we have to deal with DOPs. The challenges to algorithms in solving $\mathcal{NP}$-complete COPs with static environment are well known in terms of computational complexity [22]. Therefore, addressing $\mathcal{NP}$-complete COPs with dynamic environment, is even more challenging for algorithms because the objective is not only to locate the global optimum efficiently, but to also track it over the environmental changes [10].

The TSP becomes more 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.

Guntsch et al. [25] proposed a DTSP in which a number of cities are exchanged between the actual problem instance and a spare pool of cities. The same benchmark problem has been adapted in [35,36]. Eyckelhof and Snoek proposed the DTSP where the cost of the cities' arcs varies [21]. A similar benchmark has been adapted in [37] to represent potential traffic. Younes et al. [58] introduced a benchmark DTSP with different modes, in which each mode introduces a different dynamic. Kilby et al. [32] proposed a benchmark for the dynamic VRP (DVRP) where customer requests are revealed incrementally. The same benchmark has been adapted by Montemanni et al. [39].

3.2.1. DTSPs with random traffic jams

In this paper, we generate DTSPs via introducing the traffic factor. We assume that the cost of the link between cities i and j is $d_{ij} \times t_{ij}$, where d_{ij} is the normal travelled distance and t_{ij} is the traffic factor. Every f iterations of running an algorithm, a random number $R \in [F_L, F_U]$ is generated to represent potential traffic jams, where F_L and F_U are the lower and upper bounds of the traffic factor t_{ij}, respectively. Each link between cities i and j has a probability m to add traffic, by generating a different R to represent low, normal or high traffic jams on different roads, whereas the remaining links are set to $t_{ij}=1$, which indicates no traffic.

For example, roads with high traffic are generated by setting R values closer to F_U with a higher probability, while for roads with low traffic, a higher probability is given to generate R values closer to F_L. This type of DTSPs are denoted *random DTSPs* in this paper because previously visited environments are not guaranteed to reappear.

3.2.2. DTSPs with cyclic traffic jams

Another variation of the DTSP with traffic factors is the DTSP where the dynamic changes occur with a cyclic pattern, as illustrated in Fig. 1. In other words, previous environments are guaranteed to appear again in the future. Such environments are more realistic than random ones. For example, they represent a 24-h 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 with either low, normal, or high traffic. Then, the environment cycles among these base states in a fixed logical ring. Depending on the period of the day, environments with different traffic factors can be generated. For example, during the rush hour periods, a higher probability is given to generate R closer to F_U, whereas during evening hour periods, a higher probability is given

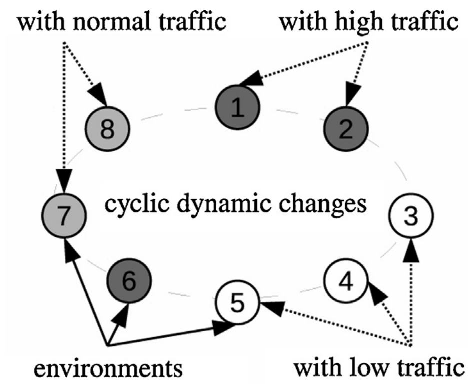

Fig. 1. Illustration of a cyclic dynamic environment with 8 states. Each node represents a different environment where white, light grey, and dark grey, represent low, normal, and high traffic jams, respectively.

to generate R closer to F_L. This type of environments are denoted as *cyclic DTSPs* in this paper.

4. Conventional ACO algorithms

4.1. $\mathcal{MM}$AS for TSPs

The first ACO algorithm, called Ant System (AS), was introduced for the static TSP [14]. Thereafter, many variations of the AS have been applied to solve difficult optimization problems [6,15,18,20]. ACO algorithms consist of a population $P(t)$ of μ ants that construct solutions, e.g., tours in the TSP, and update their trails with pheromone according to the solution quality every iteration t. 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.

One of the best performing ACO algorithm for the TSP is the $\mathcal{MAX}$-$\mathcal{MIN}$AS ($\mathcal{MM}$AS) [43,44]. More precisely, with a probability $1 - q_0$, where $0 \leq q_0 \leq 1$ is a parameter of the decision rule, an ant k in $\mathcal{MM}$AS chooses the next city j while the last city visited is i, probabilistically, as follows:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \text{ if } \quad j \in \mathcal{N}_i^k, \tag{4}$$

where τ_{ij} is the existing pheromone trail between cities i and j, $\eta_{ij} = 1/d_{ij}$ is the heuristic information available a priori, where d_{ij} is the distance between cities i and j. $\mathcal{N}_i^k$ denotes the neighbourhood of cities for ant k when the current city is i. α and β are the two parameters which determine the relative influence of τ and η, respectively. With the probability q_0, ant k chooses the next city j, with the maximum probability, which satisfies the following formula:

$$j = \underset{l \in N_i^k}{\operatorname{argmax}} \ \{\tau_{il} [\eta_{il}]^\beta\}. \tag{5}$$

The pheromone trails in $\mathcal{MM}$AS are updated by applying evaporation as follows:

$$\tau_{ij} \leftarrow (1 - \rho)\, \tau_{ij}, \forall (i, j), \tag{6}$$

where ρ is the evaporation rate which satisfies $0 < \rho \leq 1$, and τ_{ij} is the existing pheromone value. After pheromone evaporation the best ant deposits pheromone as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best}, \forall (i, j) \in T^{best}, \tag{7}$$

where $\Delta\tau_{ij}^{best} = 1/C^{best}$ is the amount of pheromone that the best ant deposits and C^{best} defines the solution quality of tour T^{best}. Since only the best ant deposits pheromone, the algorithm will quickly converge towards the best solution of the first iteration. Therefore, pheromone trail values are imposed to lower and upper limits, τ_{min} and τ_{max}, respectively, in order to avoid stagnation behaviour.

4.2. $\mathcal{MM}$AS for DTSPs

Differently from EAs, the research on ACO for DOPs is still infant. ACO algorithms are able to use knowledge from previous environments using the pheromone trails generated in the previous iterations. They also have adaptation capabilities due to pheromone evaporation [7]. For example, when a dynamic change occurs, evaporation will eliminate the pheromone trails concentrated to the old optimum. In this way, the population of ants will be flexible enough to track the moving optimum.

ACO algorithms are robust since they can accept any knowledge transferred via their pheromone trails: when the information is useful then it is considered; otherwise, it is destroyed after a few iterations, depending on the evaporation rate. However, the time required to adapt to the new environment may depend on the problem size and the magnitude of a dynamic change. When the environment changes significantly, then it will take an ACO algorithm longer to eliminate unused pheromone trails. On the other hand, pheromone evaporation may destroy information that can be used on further environments, since a bad solution in the current environment may be good in the next environments [2,21]. Several strategies have been proposed to enhance the performance of conventional ACO algorithms in dynamic environments and they are described in the next section.

5. Strategies integrated to ACO for DTSPs

Over the years, several ACO-based approaches have been introduced to address different variations of DTSPs [21,26,27,35,37]. The simplest way is to restart the algorithm after a change by re-initializing the pheromone trails with the same pheromone value [23,44] if a sufficient time to re-optimize is available. According to Jin and Branke [31] "the re-optimization time is usually short in DOPs". Furthermore, a restart strategy requires the detection of a change, which may not be applicable in some cases, e.g., dynamic environments with noise.

Guntsch and Middendorf [27] proposed local restart strategies for the DTSP, where the topology of cities changes. Similar to the global restart strategy [23,44], local restart strategies need to detect the changes, and also take into account where the change of the problem instance actually occurs. Therefore, instead of re-initializing all the pheromone trails, the pheromone trails of the cities affected by the dynamic changes, i.e., inserted or removed cities, are re-initialized. Of the two local restart strategies, the η-strategy uses heuristics-based information to modify the pheromone trails, whereas the τ-strategy uses pheromone-based information. The experimental results show that the η-strategy outperforms the τ-strategy and the global restart strategy [27].

Eyckelhof and Snoek [21] considered a different DTSP where the distances between cities change with time, representing sudden changes in the traffic. A "shaking" technique was proposed to AS where the pheromone trails are smoothed after a dynamic change and it has been compared with a traditional AS, with and without the global restart strategy, respectively. The experimental results show that their proposed approach is comparable with the other two AS variations in small problem instances.

One of the most studied approaches is the memory-based version of traditional ACO, known as the population-based ACO (P-ACO) [26]. It uses a different framework from a traditional ACO algorithm since it has a population-list (memory), denoted as $k_{long}(t)$, where the best ant of every iteration t is added. Since $k_{long}(t)$ is of a limited size (denoted as K_l), another ant needs to be removed in order to make space for a new one using the first-in-first-out policy. The pheromone trails are generated according to the solutions currently stored in $k_{long}(t)$ and there is no pheromone evaporation in P-ACO. P-ACO has been applied on a DTSP where a small portion of cities are replaced by others. When the dynamics affect the ants stored in $k_{long}(t)$, they are repaired heuristically using the keep-elitist strategy [28]. The experimental results show that P-ACO is comparable with an ACO algorithm with the global restart strategy and the η-strategy.

Moreover, variations of the P-ACO have been proposed later on where co-evolutionary approaches are integrated to the population-list, e.g., fitness sharing P-ACO (FS-PACO) and simple crowding P-ACO (SC-PACO) [1]. The main idea of these variations is to maintain the diversity in order to avoid the redundancy of the same ant within the population-list. A recent variation of P-ACO is the memetic ACO (M-ACO) where a local search based on simple and adaptive inversions is applied to the population-list in parallel with a diversity maintenance scheme based on random immigrants [36].

A good start has been made regarding the integration of immigrants schemes with ACO algorithms for permutation-encoded DOPs [35,37], considering the good performance and simplicity of immigrants schemes for DOPs and the EA characteristics of P-ACO. In [35], the RIACO and EIACO algorithms have been applied on a DTSP where the topology of cities change over time. In [37], the MIACO algorithm has been applied on the DTSP with traffic factors. The difference between RIACO, EIACO, and MIACO lies in that immigrants are generated randomly in RIACO, whereas immigrants are generated via transferring knowledge in EIACO and MIACO using the inver-over operator [29]. The algorithms have shown promising performance on some DTSPs according to our preliminary studies [35,37].

This paper extends the experiments based on two other variations of DTSPs described in Section 3. A series of different DOPs are generated to investigate the weakness and strength of RIACO, EIACO, and MIACO for solving DTSPs. This paper also carries out experiments on sensitivity analysis with respect to several important parameters, such as the immigrants replacement rate and the size of the memory used, on the performance of the investigated ACO algorithms. Moreover, the proposed algorithms are compared against several other peer ACO algorithms on the test DTSPs.

6. Proposed ACO algorithms for the DTSP

6.1. The framework of ACO with immigrants schemes

The framework of conventional ACO algorithms, e.g., $\mathcal{MM}$AS, is based on a "virtual" population of ants that construct solutions and deposit pheromone. Artificial ants move from one city to the next city probabilistically until they generate feasible solutions, and then change the pheromone information. The solutions generated by the ants are not stored in an actual population, but only in the pheromone trails, which are used by the ants of the next iteration to find better solutions. On the contrast, the framework of EAs is based on an actual population of feasible solutions which is directly transferred from one iteration to the next using selection [30,38]. Variation operators, i.e., crossover and mutation, are used to generate the new population of solutions (usually better than the previous ones).

The P-ACO algorithm is based on ants that construct solutions, where the best solution of each iteration is stored in an actual population, i.e., $k_{long}(t)$, as in EAs and is transferred directly to the next iteration. The solutions in the population are then used to update the pheromone information for the ants of the next iteration. The population-list is updated every iteration t.

The proposed framework of ACO algorithms with immigrants schemes is motivated from the EA characteristics of the P-ACO framework and the good performance of immigrants schemes in EAs for DOPs. 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, of running the algorithm, considering information from the previous environment and extra information from the immigrants schemes. Therefore, instead of using a long-term memory, i.e., $k_{long}(t)$ as in P-ACO, a short-term memory, denoted as $k_{short}(t)$, is used, where all ants stored from iteration $t-1$ are replaced by the K_s best ants of the current iteration t, where K_s is the size of $k_{short}(t)$. Moreover, a number of immigrant ants are generated and replace the worst ants in $k_{short}(t)$.

The benefits of using $k_{short}(t)$ 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 as in the P-ACO. Usually, a repair procedure is computationally expensive, and requires prior knowledge of the problem.

6.2. Construct solutions and pheromone update

The solution construction is the same as the traditional ACO, where all ants are randomly placed on cities and select the next city probabilistically, using Eq. (4). However, the pheromone update differs from both $\mathcal{MM}$AS and P-ACO. Every iteration, the pheromone matrix is associated with $k_{short}(t)$ and any change to $k_{short}(t+1)$ is reflected on the pheromone trails. For example, when the worst ants are replaced by immigrant ants, the pheromone trails of each k-th worst ant are removed, as follows:

$$\tau_{ij} \leftarrow \tau_{ij} - \Delta\tau_{ij}^{k}, \forall (i,j) \in T^{k}, \tag{8}$$

where T^k represents the tour of ant k and $\Delta\tau_{ij}^{k} = (\tau_{max} - \tau_0)/K_s$, where τ_{max} and τ_0 denote the maximum and initial pheromone values, respectively. Furthermore, the pheromone trails of each k-th immigrant ant are added, as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{k}, \forall (i,j) \in T^{k}, \tag{9}$$

where $\Delta\tau_{ij}^{k}$ and T^k are as defined in Eq. (8). This mechanism keeps the pheromone trails between a certain value τ_{min}, which is equal to τ_0, and a τ_{max} value, which can be calculated by $\tau_0 + \sum_{k=1}^{K_s} \Delta\tau_{ij}^{k}$. We have seen previously the importance to keep the pheromone trails to a certain level on the $\mathcal{MM}$AS algorithm [19,43].

From the above framework, three ACO algorithms can be instantiated based on random immigrants, elitism-based immigrants, and memory-based immigrants schemes, respectively. They are described in the following sections, respectively.

Algorithm 1. RIACO and EIACO

```
1:  $t \leftarrow 0$
2:  $P(0) \leftarrow$ InitializePopulation($\mu$)
3:  InitiliazePheromoneTrails($\tau_0$)
4:  $k_{short}(0) \leftarrow$ *empty*
5:  $x^{bs} \leftarrow$ *empty solution*
6:  **while** (termination condition *not* satisfied) **do**
7:    $P(t) \leftarrow$ ConstructAntSolutions
8:    $k_{short}(t) \leftarrow$ AddBestAnts($K_s$)
9:    **if** ($t = 0$) **then**
10:     UpdatePheromone($k_{short}(t)$)
11:   **else**
12:     if (RIACO is *selected*)
13:       $S_{ri} \leftarrow$ GenerateRandomImmigrants($r$)
14:       $k_{short}(t) \leftarrow$ ReplaceAntsWithImmigrants($S_{ri}$)
15:     **end if**
16:     **if** (EIACO is *selected*) **then**
17:       $x^{elite} \leftarrow$ FindBest($P(t-1)$)
18:       $S_{ei} \leftarrow$ *GenerateGuidedImmigrants*($x^{elite}$,$r$) using Algorithm 2
19:       $k_{short}(t) \leftarrow$ ReplaceAntsWithImmigrants($S_{ei}$)
20:     **end if**
21:     UpdatePheromone($k_{short}(t)$)
22:   **end if**
23:   $x^{ib} \leftarrow$ FindBest($P'(t)$)
24:   **if** ($f(x^{ib}) < f(x^{bs})$) **then**
25:     $x^{bs} \leftarrow x^{ib}$
26:   **end if**
27:   $t \leftarrow t+1$
28: **end while**
29: **return** $x^{bs}$
```

6.3. Random immigrants ACO (RIACO)

Random immigrants have been found to perform well with ACO for the DTSP since they maintain a certain level of diversity during the execution [35]. The principle is to introduce new randomly generated immigrant ants and replace the worst ants in $k_{short}(t)$. RIACO follows the framework described above where $k_{short}(t)$ is used instead of $k_{long}(t)$. All the ants of the current iteration replace the old ones, instead of only replacing the oldest one as in P-ACO. Therefore, when ants are removed, a negative update is made to their pheromone trails as in Eq. (8), and when new ants are added, a positive update is made to their pheromone trails as in Eq. (9). However, before the pheromone trails are updated, a set S_{ri} of $r \times K_s$ immigrants are randomly generated to replace other ants in $k_{short}(t)$, where r is called the replacement rate. The pseudo-code of RIACO is presented in Algorithm 1.

It is claimed that "the continuous adaptation of ACO algorithms makes sense only when the environmental changes of a problem are small to medium" [8]. This is due to the fact that a new environment has a high 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 this argument, we expect RIACO to perform well in quickly and significantly changing environments, since knowledge is not transferred and the diversity is generated randomly.

6.4. Elitism-based immigrants ACO (EIACO)

The RIACO algorithm works by introducing random ants into $k_{short}(t)$ as described in Section 6.3. This may increase the diversity and improve the performance of ACO algorithms in DTSPs. However, the pheromone information generated by random immigrants may misguide the ants from tracking the optimum during slight environmental changes. As a result, random immigrants may generate a high level of diversity and degrade the performance of ACO on DTSPs because of randomization.

Algorithm 2. GenerateGuidedImmigrant(x^{elite}, r)

```
1:  $x^{elite'} \leftarrow x^{elite}$
2:  $r_C \leftarrow$ SelectRandomCity($x^{elite'}$)
3:  **while** (termination condition *not* satisfied) **do**
4:    **if** ($rand[0.0, 1.0] \leq 0.02$) **then**
5:      $r_{C'} \leftarrow$ SelectNewRandomCity($x^{elite'}$)
6:    **else**
7:      $x^r \leftarrow$ SelectRandomAnt($P(t)$)
8:      $r_{C'} \leftarrow$ NextCity($r_C + 1, x^r$) $\in x^r$
9:    **end if**
10:   **if** ($r_{C'} = r_C + 1 \in x^{elite'}$ *or* $r_{C'} = r_C - 1 \in x^{elite'}$) **then**
11:     **break**
12:   **else**
13:     Inversion($r_C + 1, r_{C'}$) $\in x^{elite'}$
14:     $r_C \leftarrow r_{C'}$
15:   **end if**
16: **end while**
17: **return** $x^{elite'}$ // guided immigrant generated
```

In order to generate guided diversity, EIACO was proposed to address the DTSP by transferring knowledge from the previous environment [35]. For each iteration t, within EIACO, the elite ant from the previous environment, i.e., the best ant from $P(t-1)$, is used as the base to generate a set S_{ei} of $r \times K_s$ elitism-based immigrants using the inver-over operator [29] presented in Algorithm 2, where two cities are selected and the segment between them is reversed. The pseudo-code of EIACO is also shown in Algorithm 1.

In cases where the changing environments are similar, e.g., the magnitude of change is small, and when the population has sufficient time to converge into a good solution in the previous environment, EIACO may be beneficial. Transferring the knowledge gained from the previous environment, using guided immigrants, to the pheromone trails of the new environment will guide the population of ants to promising areas of the new environment quickly. However, if too much information is transferred, the run may start near a local optimum, and will get stuck there. Therefore, in some cases with slightly changing environments, EIACO may not perform well.

6.5. Memory-based immigrants ACO (MIACO)

MIACO is a generalized version of EIACO since not only the best ant from the previous environment, but the best ant among several previous environments is considered as the base to generate immigrants in MIACO. The only difference between MIACO and EIACO lies in that MIACO uses both $k_{short}(t)$ and $k_{long}(t)$, 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 exist 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 the best-so-far ant is better. Note that the update strategy of $k_{long}(t)$ in MIACO is different from that in P-ACO regarding which ant to replace, since in MIACO the most similar memory updating strategy is used [8], whereas in the default P-ACO, the new ant replaces the oldest one. In MIACO, a metric of how close ant p is to ant q is used and is defined as follows:

$$M(p, q) = 1 - \frac{c_{E_{pq}}}{n}, \tag{10}$$

where $c_{E_{pq}}$ is defined as the number of common edges between the solution of ant p and the solution of ant q, and n is the number of cities. A value $M(p, q)$ closer to 0 means that the ants are closer since they are more similar [1].

Apart from which ant is replaced in $k_{long}(t)$, 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 occurs in order to store useful solutions from different environments. Therefore, for each iteration t within MIACO, the ants in $k_{long}(t)$ are re-evaluated in order to be valid with the new environment and to detect an environmental change. Note that an environmental change is detected if there is a change in the total cost of $k_{long}(t)$ (if applicable). Then, the best ant from $k_{long}(t)$ is selected and used as the base to generate a set S_{mi} of $r \times K_s$ memory-based immigrants using Algorithm 2. The pseudo-code of MIACO is presented in Algorithm 3.

Algorithm 3. MIACO

```
1:  $t \leftarrow 0$
2:  $P(0) \leftarrow$ InitializePopulation($\mu$)
3:  InitializePheromoneTrails($\tau_0$)
4:  $k_{short}(0) \leftarrow$ *empty*
5:  $k_{long}(0) \leftarrow$ InitializeRandomly($K_l$)
6:  $t_M \leftarrow rand[5, 10]$
7:  $x^{bs} \leftarrow$ *empty solution*
8:  **while** (termination condition *not* satisfied) **do**
9:    $P(t) \leftarrow$ ConstructAntSolutions
10:   $k_{short}(t) \leftarrow$ AddBestAnts($K_s$)
11:   **if**($t = t_M$ *or* dynamic change is *detected*) **then**
12:     *UpdateMemory*($k_{long}(t)$) using Algorithm 4
13:     $t_M \leftarrow t + rand[5, 10]$
14:   **end if**
15:   **if** ($t = 0$) **then**
16:     UpdatePheromone($k_{short}(t)$)
17:   **else**
18:     $x^{elite} \leftarrow$ FindBest($k_{long}(t)$)
19:     $S_{mi} \leftarrow$ *GenerateGuidedImmigrants*($x^{elite}$,$r$) using Algorithm 2
20:     $k_{short}(t) \leftarrow$ ReplaceAntsWithImmigrants($S_{mi}$)
21:     UpdatePheromone($k_{short}(t)$)
22:   **end if**
23:   $x^{ib} \leftarrow$ FindBest($P'(t)$)
24:   **if** ($f(x^{ib}) < f(x^{bs})$) **then**
25:     $x^{bs} \leftarrow x^{ib}$
26:   **end if**
27:   $t \leftarrow t + 1$
28:   $k_{long}(t) \leftarrow k_{long}(t-1)$
29: **end while**
30: **return** $x^{bs}$
```

However, the update does not depend only on the detection of dynamic changes since in some real-world applications it is not easy or possible to detect changes, as discussed previously. For example, in DTSPs, noise may be added in every iteration of the algorithm apart from the traffic factor, which may also indicate environmental changes using the detection mechanism described above. Hence, the algorithm will not be able to distinguish whether the change of the fitness in a solution is because of noise or an environmental change. The change detection mechanism will not work properly because it will detect changes in every iteration due to the noise. Therefore, instead of updating $k_{long}(t)$ only in a fixed time interval, e.g., every f iterations, which is dependent on the dynamic changes, $k_{long}(t)$ is also updated in a dynamic pattern as presented in Algorithms 3 and 4. On every update of $k_{long}(t)$, a random number $R \in [5, 10]$ is generated, which indicates the next update time. For example, if the memory is updated at iteration t, the next update will occur at iteration $t_M = t + R$ [53,57] (if no change is detected before iteration t_M).

Algorithm 4. UpdateMemory($k_{long}(t)$)

```
1:  **if** ($t = t_M$) **then**
2:    $x^{mb} \leftarrow$ FindBest($P(t)$)
3:  **end if**
4:  **if** (dynamic change is *detected*) **then**
5:    $x^{mb} \leftarrow$ FindBest($P(t-1)$)
6:  **end if**
7:  **if** (still any random ant in $k_{long}(t)$) **then**
8:    ReplaceRandomWithBest($x^{mb}$,$k_{long}(t)$)
9:  **else**
10:   $x^{cm} \leftarrow$ FindClosest($x^{mb}$,$k_{long}(t)$)
11:   **if** ($f(x^{mb}) < f(x^{cm})$) **then**
12:     $x^{cm} \leftarrow x^{mb}$
13:   **end if**
14: **end if**
```

MIACO inherits the advantages of the memory scheme to guide the population directly to an old environment already visited and the guided immigrants scheme to maintain diversity of the population in order to avoid the stagnation behaviour of ACO algorithms. It is very important to store different solutions in $k_{long}(t)$ which represent different environments that are useful in the future. The key idea behind MIACO is to provide guided diversity, using guided immigrants, into the pheromone trails in order to avoid the disruption of the optimization process [52].

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

7. Experimental study

7.1. Experimental setup

In order to investigate the effect of immigrants schemes on ACO algorithms for the DTSP, several experiments are carried out in this study. All algorithms were tested on the DTSP instances that are generated from three stationary benchmark TSP instances taken from TSPLIB[1], i.e., kroA100, kroA150, and kroA200, which represent small, medium, and larger scale problem instances in this paper, respectively. Our implementation follows the guidelines of the ACOTSP[2] application. Moreover, the proposed algorithms, i.e., RIACO, EIACO, and MIACO, are compared with other peer ACO algorithms described in Section 5.

Using the proposed methods described in Section 3, we have generated two kinds of DTSPs, with random and cyclic traffic factors, respectively, with $F_L = 0$ and $F_U = 5$. In cyclic DTSPs, three cyclic states are used. For both types of DTSPs, the value of f was set to 5 and 100, 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, twelve dynamic test DTSP instances, i.e., three values of $f \times$ four values of m, were generated from each static TSP instance. Therefore, in order to systematically analyse the adaptation and searching capabilities of each algorithm on the DTSP, 36 dynamic test cases are used, i.e., three problem instances $\times$ twelve cases each, for each type of DTSPs, i.e., DTSPs with random and cyclic traffic factors.

For each algorithm on a DTSP instance, $N = 30$ independent runs were executed on the same environmental changes. The algorithms were executed for $G = 1000$ iterations and one observation was taken on each iteration. The overall offline performance of an algorithm on a DTSP instance is defined as follows [31]:

$$\bar{P}_{OFF} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N}P_{ij}^{*}\right), \qquad (11)$$

where P_{ij}^{*} defines the tour cost of the best ant of iteration i of run j.

7.2. Parameter settings

Some of the parameters of the studied algorithms are chosen from our preliminary experiments [35,37] and some of them are taken from the literature [1,25]. We have set $\beta \in \{2, 5, 8\}$ and $q_0 \in \{0.0, 0.5, 0.9\}$ and kept the parameters that provide the best results for the algorithms. Then, these parameter settings have been used to optimize some key parameters, i.e., ρ of $\mathcal{MM}$AS in

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

Table 2
Parameters used for the investigated algorithms in the experiments.

Alg.	μ	α	β	q_0	K_l	K_s
			Proposed ACO algorithms			
RIACO	28	1	5	0.0	–	6
EIACO	28	1	5	0.0	–	6
MIACO	25	1	5	0.0	3	6
			Other peer ACO algorithms			
$\mathcal{MM}$AS [43]	28	1	5	0.0	–	–
P-ACO [26]	28	1	5	0.9	3	0
FS-PACO [1]	28	1	5	0.5	0	8
SC-PACO [1]	20	1	5	0.5	8	0
M-ACO [36]	20	1	5	0.9	3	0

Section 7.2.1, K_s in Section 7.2.2, K_l in Section 7.2.3 of MIACO, and r of RIACO, EIACO and MIACO in Section 7.4. The parameter settings of the proposed algorithms used in the basic experiments in Section 7.3, that have been determined in the above way, and of other existing peer ACO algorithms used in the experiments in Section 7.7 are presented in Table 2. The existing and the proposed algorithms are described in Sections 5 and 6, respectively.

The population size of M-ACO is set to $\mu - ls$, where $ls = 8$ determines the steps of the local search improvements that count as evaluations. The population size of the remaining ACO algorithms depends on the size of K_l since the ants stored in $k_{long}(t)$ are used as detectors, and they are re-evaluated in every iteration t to detect environmental changes in order to update $k_{long}(t)$. Therefore, the population size of the algorithms that use $k_{long}(t)$ is set to $\mu - K_l$, e.g., MIACO and SC-PACO. For P-ACO and M-ACO there is no need to re-evaluate $k_{long}(t)$ because the update policy of these algorithms is not depended on the solution quality. Furthermore, the evaporation rate of $\mathcal{MM}$AS is set to $\rho = 0.6$.

7.2.1. Experimental study on the effect of the evaporation rate for $\mathcal{MM}$AS

For the results shown in Table 3, $\mathcal{MM}$AS is applied to slowly changing environments with different magnitudes of change, and the evaporation rate is optimized among $\rho \in \{0.02, 0.2, 0.4, 0.6, 0.8\}$. As discussed in Section 4.2, the pheromone evaporation is an important mechanism that enables ACO to adapt in dynamic changes.

It can be observed that when the evaporation rate of $\mathcal{MM}$AS is $\rho = 0.02$ it has the worst results in all dynamic test cases, whereas

Table 3
Offline performance of $\mathcal{MM}$AS with different evaporation rates on random DTSPs, where the best result among different ρ settings on the same problem instance is highlighted in bold font.

$f = 100$, $m \Rightarrow$	0.1	0.25	0.5	0.75
Alg. &Inst.		kroA100		
$\mathcal{MM}$AS($\rho = 0.02$)	24445.4	26987.6	34702.2	47587.1
$\mathcal{MM}$AS($\rho = 0.2$)	23372.7	25759.4	32947.6	45133.6
$\mathcal{MM}$AS($\rho = 0.4$)	**23254.5**	25573.6	**32766.3**	44748.2
$\mathcal{MM}$AS($\rho = 0.6$)	23261.6	25564.1	32795.6	**44666.9**
$\mathcal{MM}$AS($\rho = 0.8$)	23260.2	**25547.9**	32821.3	44819.2
Alg. &Inst.		kroA150		
$\mathcal{MM}$AS($\rho = 0.02$)	31686.2	35607.3	43066.7	59458.6
$\mathcal{MM}$AS($\rho = 0.2$)	30058.7	33984.6	41006.5	56617.8
$\mathcal{MM}$AS($\rho = 0.4$)	29830.2	33656.3	40636.8	**56061.1**
$\mathcal{MM}$AS($\rho = 0.6$)	**29792.4**	**33499.8**	**40569.8**	56263.3
$\mathcal{MM}$AS($\rho = 0.8$)	29835.1	33595.1	40681.0	56371.8
Alg. &Inst.		kroA200		
$\mathcal{MM}$AS($\rho = 0.02$)	35971.3	39712.1	48717.9	69093.5
$\mathcal{MM}$AS($\rho = 0.2$)	33996.8	37998.9	46556.8	66113.6
$\mathcal{MM}$AS($\rho = 0.4$)	33586.6	37728.6	**46054.1**	**65225.7**
$\mathcal{MM}$AS($\rho = 0.6$)	**33495.1**	**37588.2**	46109.5	65408.8
$\mathcal{MM}$AS($\rho = 0.8$)	33518.4	37699.1	46403.8	65787.9

Table 4
Offline performance for the proposed framework with different short-term memory sizes on random DTSPs but without immigrants generated, where the best result among different K_s settings on the same problem instance is highlighted in bold font.

$f = 100$, $m \Rightarrow$	0.1	0.25	0.5	0.75
Alg. &Inst.		kroA100		
$K_s = 1$	**23327.1**	**25664.9**	33122.1	45576.5
$K_s = 6$	23499.9	25769.1	**32638.4**	**44472.5**
$K_s = 12$	24199.0	26661.4	33364.4	45076.2
$K_s = 28$	26462.8	29071.8	36443.4	49346.2
Alg. &Inst.		kroA150		
$K_s = 1$	30297.9	34311.2	41494.7	57547.8
$K_s = 6$	**29957.0**	**33590.8**	**40311.1**	**55332.4**
$K_s = 12$	30850.3	34540.2	41089.6	55795.3
$K_s = 28$	33681.9	37393.6	44742.5	60041.1
Alg. &Inst.		kroA200		
$K_s = 1$	34368.5	38497.1	47305.4	67039.0
$K_s = 6$	**33789.2**	**37347.2**	**45441.3**	**64139.8**
$K_s = 12$	35080.0	38718.8	46230.1	64398.4
$K_s = 28$	38050.7	41942.4	50244.6	69031.0

$\rho = 0.02$ is the recommended evaporation rate in static TSPs [20, p. 71]. This is natural because an extremely low evaporation rate corresponds to slow adaptation. More precisely, the pheromone trails of the previous environment that bias the ants towards non-promising areas in the search space are eliminated slowly. As the evaporation rate increases to 0.6, the solutions quality improves, whereas when it is higher than 0.6, the solution quality is usually degraded or remains similar. This is because an extremely high evaporation rate may eliminated useful information from the pheromone trails of the previous environment, and the knowledge transferred is destroyed quickly.

7.2.2. Experimental study on the effect of the short-term memory size for RIACO, EIACO and MIACO

For the results shown in Table 4, the proposed framework with the immigrant ants scheme switched off is applied to slowly changing environments with different magnitudes, where the short-term memory size is set to $K_s \in \{1, 6, 12, 28\}$, respectively. As discussed in Section 6.2, the proposed framework does not depend on pheromone evaporation as with $\mathcal{MM}$AS, but it re-generates the pheromone trails every iteration from the solutions stored in $k_{short}(t)$.

It can be observed that when $K_s = 6$ the performance of the algorithm is improved, probably because the pheromone trails get more information from the previous environment to adapt to a new environment. However, when an extreme value is given to K_s, e.g., when K_s is close to the population size μ, then the algorithm has no selection with respect to the best ants. This is because all the ants in the framework deposit a constant amount of pheromone and, thus, no effort is given to the best ants.

If we compare the results of the proposed framework in Table 4 against the results of $\mathcal{MM}$AS in Table 3, there is an interesting observation. The performance of $\mathcal{MM}$AS($\rho = 0.6$) is better than the performance of the proposed framework with $K_s = 6$ when $m = 0.1$ and $m = 0.25$, while the performance of the latter is better than that of the former when $m = 0.5$ and $m = 0.75$. For example, on kroA150 with $m = 0.1$ the performance of $\mathcal{MM}$AS($\rho = 0.6$) and $K_s = 6$ is $\overline{P}_{OFF} = 29792.4$ and $\overline{P}_{OFF} = 29957.0$, respectively, whereas on the same problem instance with $m = 0.75$ the performance of the two algorithms is $\overline{P}_{OFF} = 56263.3$ and $\overline{P}_{OFF} = 55332.4$, respectively. This is because if the dynamic change is severe, then it will take more time for $\mathcal{MM}$AS to eliminate unused pheromone trails, whereas the proposed framework removes the previous pheromone trails directly. This supports our claim in Section 4.2 that the performance of $\mathcal{MM}$AS is degraded as the magnitude of change increases.

Table 5
Offline performance of the proposed MIACO with different long-term memory sizes on cyclic DTSPs, where the best result among different K_l settings on the same problem instance is highlighted in bold font.

f = 50, m⇒	0.1	0.25	0.5	0.75
Alg. &Inst.	kroA100 (3 cyclic states)			
K_l = 1	**23673.9**	**26779.8**	32197.9	38740.6
K_l = 3	23853.9	26832.3	**32114.7**	**38634.0**
K_l = 6	23985.0	26935.4	32336.5	38901.3
K_l = 9	24242.7	27130.2	32506.3	39097.4
Alg. &Inst.	kroA100 (6 cyclic states)			
K_l = 1	**24037.1**	**26703.3**	31752.4	41424.5
K_l = 3	24137.4	26772.2	**31639.1**	**41285.3**
K_l = 6	24307.7	26901.3	31867.8	41596.0
K_l = 9	24513.5	27109.7	31994.6	41833.1
Alg. &Inst.	kroA100 (10 cyclic states)			
K_l = 1	**24085.6**	**26682.7**	31572.7	41623.0
K_l = 3	24136.1	26772.9	**31483.1**	**41506.8**
K_l = 6	24340.6	26901.8	31718.7	41777.4
K_l = 9	24570.9	27087.1	31884.7	42002.3

7.2.3. *Experimental study on the effect of the long-term memory size for MIACO*

For the results shown in Table 5, MIACO is applied to cyclic dynamic environments with different magnitudes, where the long-term memory size is set to $K_l \in \{1, 3, 6, 9\}$. The population size of MIACO depends on the size K_l and is set as $\mu - K_l$ (see Section 7.2 for more details). Considering the description of MIACO in Section 6.5, each solution stored in $k_{long(t)}$ represents one previous environment so that it can be used when the same environment re-appears. Hence, an ideal K_l value might be equal to the number of states in the cyclic environment.

However, it can be observed that when K_l = 1, MIACO performs better when the environment changes slightly whereas when K_l = 3, MIACO performs better when the environment changes significantly. This is because when m = 0.1 and m = 0.25 the environments are more likely to be similar and there is no need to store many solutions in $k_{long}(t)$. Moreover, the performance of MIACO when K_l = 3 is the same in all problems with different numbers of cyclic states. This is because a solution stored in $k_{long}(t)$ may be useful for several environments. This way, fitness evaluations are not wasted from the decreased population, i.e., when K_l = 9.

7.3. *Experimental study on comparing the proposed ACO algorithms on different DTSPs*

The basic experimental results regarding the offline performance of RIACO, EIACO and MIACO algorithms in both DTSPs with random and cyclic traffic factors are presented in Table 6. The corresponding Wilcoxon sum-rank results at a 0.05 level of significance are presented in Table 7. Moreover, to better understand the dynamic behaviour of algorithms, the offline performance against 100 iterations is plotted in Fig. 2 for random DTSPs with f = 5 and m = 0.1 and m = 0.75, and the offline performance against the first 1000 iterations is plotted in Fig. 3 for cyclic DTSPs with f = 100 and m = 0.1 and m = 0.75. From the experimental results, several observations regarding the weaknesses and strengths can be made by comparing the proposed algorithms.

First, RIACO significantly outperforms EIACO in almost all random and cyclic DTSPs, when f = 5 and m = 0.75; see the comparisons of EIACO ⇔ RIACO in Table 7. This validates our expectation that random immigrants perform well in quickly and significantly changing environments. However, in the remaining dynamic test cases RIACO is outperformed by EIACO. This is because RIACO increases diversity randomly via random immigrants and there is a high risk of too much randomization that may disturb the re-optimization process.

Second, EIACO significantly outperforms RIACO in almost all random and cyclic DTSPs, when f = 100 and when m = 0.1, m = 0.25 and m = 0.5; see the comparisons of EIACO ⇔ RIACO in Table 7. This validates our expectation that elitism-based immigrants perform well in slightly and slowly changing environments. This is because the knowledge transferred from the pheromone trails of the previous environment by the elitism mechanism requires similar environments or enough time to express its effect and speed up the re-optimization process. However, in cyclic DTSPs as the value of m increases the performance of EIACO is degraded, even when f = 100, whereas the performance of MIACO is improved. The same observation was made in Section 7.2.3. This is because MIACO uses the best solution from $k_{long}(t)$, whereas EIACO uses the best solution from the previous environment. In this way, MIACO has a variety of best solutions to choose, whereas EIACO is limited to the choice of the best solution from $P(t-1)$.

Third, MIACO has a similar performance with EIACO, when compared with RIACO in all random DTSPs; see the comparisons

Table 6
Experimental results regarding the offline performance of the algorithms on random and cyclic DTSPs, where the best result among algorithms on the same problem instance is highlighted in bold font.

Alg. &Inst.	kroA100				kroA150				kroA200			
	DTSPs with random traffic factor											
f = 5, m⇒	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	26557.8	30420.4	38252.7	**53471.6**	33816.5	38512.4	48133.7	**66340.4**	38535.5	43914.6	55048.9	**76008.3**
EIACO	**26100.0**	**30258.0**	**38166.9**	53491.6	**33376.1**	**38272.3**	**48053.8**	66428.3	**37926.2**	**43576.3**	**54922.2**	76103.4
MIACO	26198.9	30341.5	38312.4	53728.9	33511.3	38382.4	48243.0	66674.8	38050.7	43677.9	55108.7	76355.5
f = 100, m⇒	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	23635.8	25846.7	32876.3	44905.7	30343.8	33976.4	40704.9	56085.0	34203.1	37766.3	46036.8	65169.9
EIACO	**23417.2**	**25660.5**	**32576.1**	**44339.0**	**29892.6**	**33280.7**	**39994.4**	**54963.4**	**33496.6**	**37072.2**	**45106.1**	**63347.5**
MIACO	23398.7	25736.8	32687.5	44458.1	29893.0	33443.3	40175.3	55175.1	33576.9	37236.1	45206.7	63781.2
	DTSPs with cyclic traffic factor											
f = 5, m⇒	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	25757.7	29990.3	37189.3	45013.2	33389.7	37091.3	42240.5	56595.2	36192.8	41550.4	49383.7	61938.5
EIACO	**23991.4**	**29530.6**	37213.1	45013.2	**30671.2**	**36228.3**	**41787.5**	56720.6	33169.1	**40329.2**	**48421.4**	61754.9
MIACO	24106.7	29586.5	**37001.2**	**44630.3**	30767.2	36337.4	41829.0	**56243.3**	**33164.1**	40375.5	48565.4	**61524.3**
f = 100, m⇒	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	23980.4	26401.9	31072.9	37717.3	30245.7	32323.3	36056.6	47116.3	33641.2	35914.9	42646.5	52493.2
EIACO	**23220.3**	26061.0	30988.1	37486.1	**29558.5**	**32012.2**	35761.9	46395.4	**32381.9**	35516.8	**41779.5**	51617.5
MIACO	23272.8	**26031.9**	**30850.6**	**37361.5**	29732.8	32035.5	**35745.8**	**46251.5**	32558.8	**35450.6**	41842.3	**51594.0**

Table 7
Statistical test results regarding the offline performance of the algorithms on random and cyclic DTSPs, where "+" or "−" indicates that the first algorithm is significantly better or the second algorithm is significantly better, respectively, and "∼" indicates no significance.

Alg. &Inst.	kroA100				kroA150				kroA200			
					DTSPs with random traffic factor							
$f = 5$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
EIACO⇔RIACO	+	+	+	∼	+	+	+	−	+	+	+	−
MIACO⇔RIACO	+	+	−	−	+	+	−	−	+	+	−	−
MIACO⇔EIACO	−	−	−	−	−	−	−	−	−	−	−	−
$f = 100$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
EIACO⇔RIACO	+	+	+	+	+	+	+	+	+	+	+	+
MIACO⇔RIACO	+	+	+	+	+	+	+	+	+	+	+	+
MIACO⇔EIACO	∼	∼	−	−	∼	−	−	−	∼	−	∼	−
					DTSPs with cyclic traffic factor							
$f = 5$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
EIACO⇔RIACO	+	+	∼	∼	+	+	+	−	+	+	+	+
MIACO⇔RIACO	+	+	+	+	+	+	+	+	+	+	+	+
MIACO⇔EIACO	−	−	+	+	∼	−	∼	+	∼	∼	−	+
$f = 100$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
EIACO⇔RIACO	+	+	+	+	+	+	+	+	+	+	+	+
MIACO⇔RIACO	+	+	+	+	+	+	+	+	+	+	+	+
MIACO⇔EIACO	∼	∼	+	+	−	∼	∼	+	−	∼	∼	∼

of MIACO ⇔ RIACO. This is due to the same reasons discussed above for EIACO, since both EIACO and MIACO generate guided immigrants. However, MIACO is outperformed by EIACO in all random DTSPs, whereas EIACO is outperformed by MIACO in most cyclic DTSPs (expect when $m = 0.1$); see the comparisons of MIACO ⇔ EIACO in Table 7. This validates our expectation that EIACO performs well when the changing environments are similar and MIACO performs well when the changing environments reappears. The reason why EIACO is effective in slightly dynamic environments is explained above. The reason why MIACO is effective in cyclic dynamic environments lies in 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. In some cases, e.g., when $f = 5$, MIACO is outperformed by EIACO. This is because that when the environment changes quickly, the best solution stored in the memory may not be able to track (or represent) the optimum of the current environment and hence may misguide the memory-based immigrants to a non-promising area.

7.4. *Experimental study on the effect of the replacement rate for random and guided immigrants*

In order to investigate the effect of the immigrants replacement rate, further experiments were carried out on the same problem instances with the same parameters used before but with different immigrant replacement rates, i.e., $r \in \{0.0, 0.2, 0.4, 0.8\}$. Note that when $r = 0.0$, it means that no immigrants are generated and hence

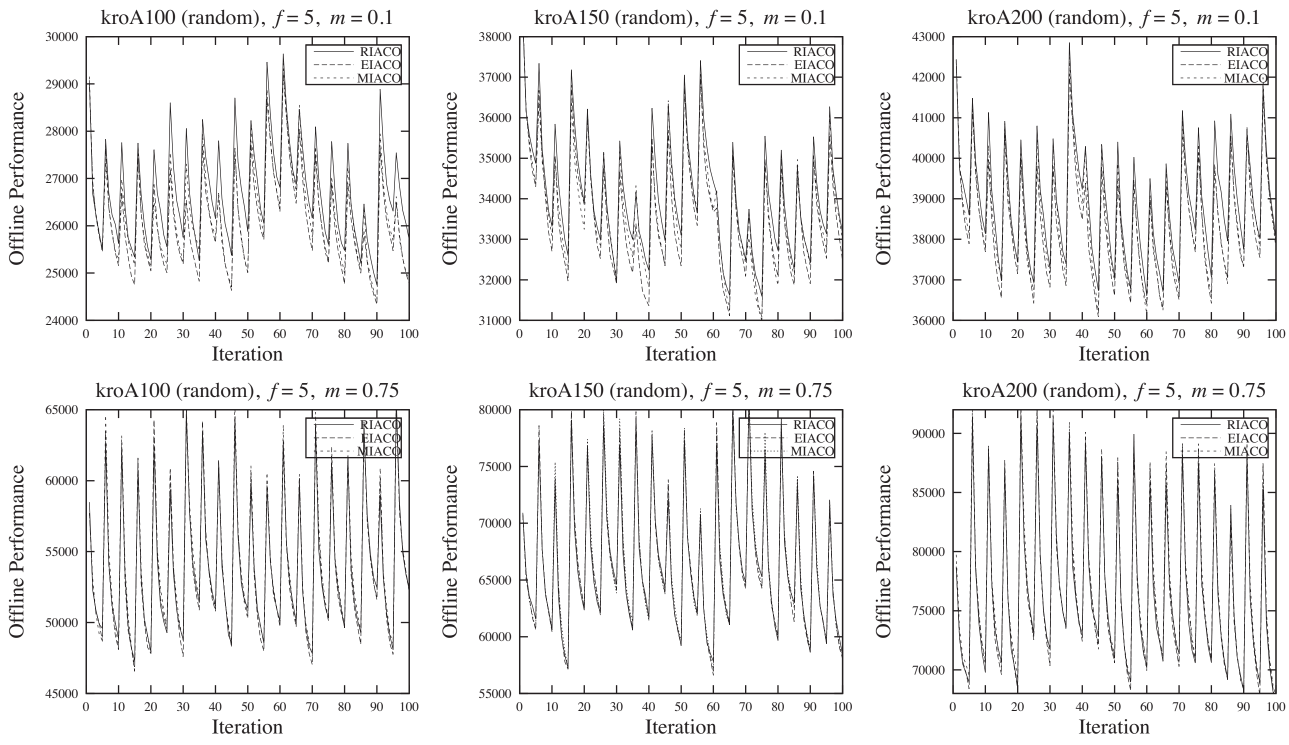

Fig. 2. Dynamic behaviour of the proposed investigated ACO algorithms on random DTSPs.

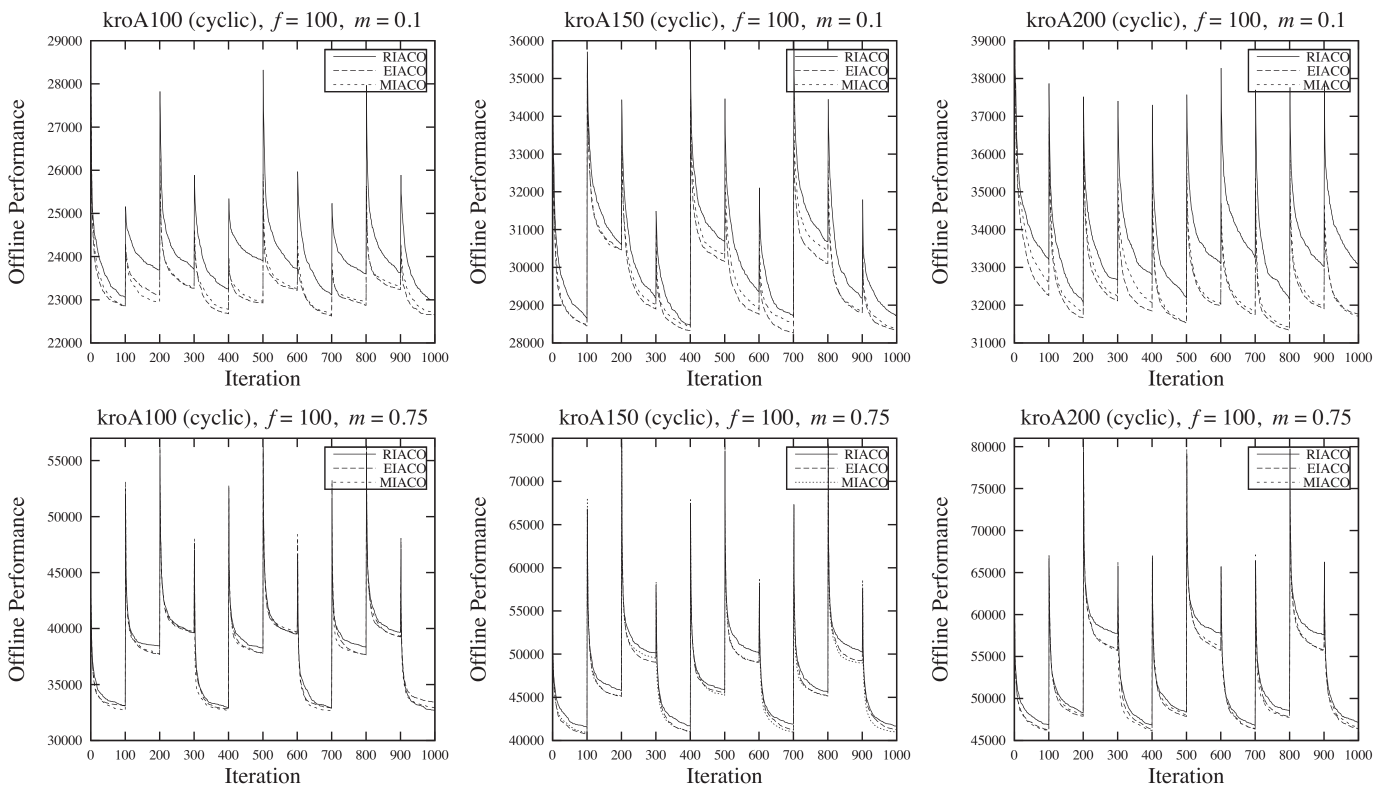

Fig. 3. Dynamic behaviour of the proposed investigated ACO algorithms on cyclic DTSPs.

the performance of EIACO($r = 0.0$) is the same as the performance of $K_s = 6$ in Table 4. Also note that here RIACO and EIACO (or MIACO) are applied on the DTSPs where the algorithms respectively showed good performance according to the basic experiments in Section 7.3. That is, RIACO is applied in quickly changing environments, while EIACO is applied in slowly changing environments. The results are shown in Tables 8 and 9, respectively.

From Table 8, it can be seen that random immigrants improve the performance of ACO when $m = 0.75$ and confirms that RIACO performs well on fast and significantly changing environments. Furthermore, when $m = 0.1$, $m = 0.25$ and $m = 0.5$, RIACO ($r = 0.0$) performs better, which shows that random immigrants disturb the optimization process because of too much randomization and furthermore validates our expectation mentioned before.

From Table 9, it can be seen that guided immigrants improve the performance of ACO in all cases when the replacement rate is $0.0 < r \leq 0.4$, and confirms that EIACO performs well on slowly changing environments. Furthermore, when the replacement rate is $r = 0.8$, the performance is usually degraded and sometimes is even worse than EIACO ($r = 0.0$). This shows that too much knowledge transferred may degrade the performance in DTSPs. This is because the re-optimization process in a new environment may start from a local optimum (or a near local optimum) location and get the algorithm stuck there, which furthermore supports our claim in Section 6.4.

Table 8
Offline performance for RIACO with different immigrants replacement rates on random DTSPs, where the best result among different r settings on the same problem instance is highlighted in bold font.

$f = 5$, $m \Rightarrow$	0.1	0.25	0.5	0.75
Alg. &Inst.		kroA100		
RIACO($r = 0.0$)	**26178.0**	**30059.7**	**37906.4**	53899.5
RIACO($r = 0.2$)	26373.6	30227.1	38060.5	53616.7
RIACO($r = 0.4$)	26557.8	30420.4	38252.7	**53471.6**
RIACO($r = 0.8$)	27401.3	31177.1	39076.6	54474.6
Alg. &Inst.		kroA150		
RIACO($r = 0.0$)	**33230.1**	**37929.1**	**47641.4**	66835.9
RIACO($r = 0.2$)	33517.7	38219.4	47885.3	66564.5
RIACO($r = 0.4$)	33816.5	38512.4	48133.7	**66340.4**
RIACO($r = 0.8$)	34989.8	39550.3	49169.2	67631.4
Alg. &Inst.		kroA200		
RIACO($r = 0.0$)	**37783.6**	**43147.7**	**54345.4**	76395.8
RIACO($r = 0.2$)	38175.8	43566.0	54712.6	76265.4
RIACO($r = 0.4$)	38535.5	43914.6	55048.9	**76008.3**
RIACO($r = 0.8$)	39767.2	45080.5	56228.9	77399.6

Table 9
Offline performance for EIACO with different immigrants replacement rates on random DTSPs, where the best result among different r settings on the same problem instance is highlighted in bold font.

$f = 100$, $m \Rightarrow$	0.1	0.25	0.5	0.75
Alg. &Inst.		kroA100		
EIACO($r = 0.0$)	23499.9	25769.1	32638.4	44472.5
EIACO($r = 0.2$)	23417.4	**25630.9**	32596.5	**44222.9**
EIACO($r = 0.4$)	23417.2	25660.5	**32576.1**	44339.0
EIACO($r = 0.8$)	**23334.2**	25782.1	33069.1	45085.4
Alg. &Inst.		kroA150		
EIACO($r = 0.0$)	29957.0	33590.8	40311.1	55332.4
EIACO($r = 0.2$)	**29791.9**	33390.0	40042.5	55010.9
EIACO($r = 0.4$)	29892.6	**33280.7**	**39994.4**	**54963.4**
EIACO($r = 0.8$)	30018.0	33742.1	40812.7	56427.4
Alg. &Inst.		kroA200		
EIACO($r = 0.0$)	33789.2	37347.2	45441.3	64139.8
EIACO($r = 0.2$)	**33444.6**	37142.6	**45062.8**	63653.0
EIACO($r = 0.4$)	33496.6	**37072.2**	45106.1	**63347.5**
EIACO($r = 0.8$)	33656.4	37646.8	46063.6	64830.6

7.5. Experimental study on the population diversity of the proposed ACO algorithms

In order to investigate the effect of random, elitism-based, and memory-based immigrants on the population diversity of ACO, we calculate the total population diversity as follows:

$$\overline{T}_{DIV} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N} DIV_{ij}\right), \tag{12}$$

where N is the number of runs, G is the number of iterations, and DIV_{ij} is the diversity of the population in iteration i of run j. For the DTSPs, DIV_{ij} can be calculated as follows:

$$DIV_{ij} = \frac{1}{\mu(\mu-1)}\sum_{p=1}^{\mu}\sum_{q \neq p}^{\mu} M(p,q), \tag{13}$$

where μ is the size of population, and $M(p, q)$ is the similarity metric between ant p and ant q as defined in Eq. (10). The total diversity results of the dynamic test cases with $f = 100$ are presented in Fig. 4 for random DTSPs. The corresponding experimental results on cyclic DTSPs show similar observations and hence are not presented here.

Generally speaking, it can be observed that RIACO maintains higher diversity than the remaining algorithms, whereas EIACO and MIACO maintain similar diversity level in most cases. This shows that random immigrants generate higher diversity than guided immigrants.

Comparing the results of the offline performance in Table 6 against the results of the diversity in Fig. 4, it can be observed that ACO algorithms that maintain high diversity levels do not always achieve better performance than other ACO algorithms for the DTSP.

7.6. Experimental study on the effect of traffic factor bounds on RIACO, EIACO and MIACO performance

In the basic experiments above, the traffic factor bounds F_L and F_U are set to 0 and 5, respectively. In order to investigate the effect of the range of the traffic bounds on the performance of ACO algorithms, further experiments are carried out in cyclic DTSPs with $f = 100$ and $m = 0.1$ and $m = 0.75$. The upper bound of the traffic factor F_U, which determines the range of the traffic factor, is set to $F_U \in \{1, 5, 10, 20\}$, whereas the lower bound F_L remains 0. The remaining experimental settings are the same as in the basic experiments. The experimental results of $f = 100$ and $m = 0.1$ are plotted in Fig. 5. The corresponding experimental results of $f = 100$ and $m = 0.75$ show similar observations and hence are not presented here.

From Fig. 5 it can be observed that as the F_U increases the performance of ACO algorithms is degraded on DTSPs with the same f and m values. The performance of ACO algorithms is affected by the range of the traffic factor and the effect differs with ACO algorithms. More precisely, the performance of MIACO, which uses memory, is degraded significantly as the F_U increases. For example, on the kroA100 with $f = 100$ and $m = 0.1$, the performance of MIACO is degraded from $\overline{P}_{OFF} = 22489.5$ at $F_U = 1$, to $\overline{P}_{OFF} = 23398.7$ at $F_U = 5$, to $\overline{P}_{OFF} = 23773.6$ at $F_U = 10$, to $\overline{P}_{OFF} = 23869.5$ at $F_U = 20$. This effect is natural since the higher range of traffic factor means the environment will return to a base state less accurately, which makes the memory scheme less efficient.

Furthermore, the performance of EIACO is improved against MIACO as the F_U value increases. For example, on the kroA100 with $f = 100$ and $m = 0.1$, the performance improvement of EIACO over MIACO, i.e., $\overline{P}_{OFF}$(MIACO)-$\overline{P}_{OFF}$(EIACO), increases from 23272.8 − 23220.3 = 52.5 at $F_U = 5$ to 23869.5 − 23754.4 = 115.1 at $F_U = 20$. However, the range of the traffic factor does not affect the ranking of the algorithms when $F_U \geq 5$, but it does affect the ranking

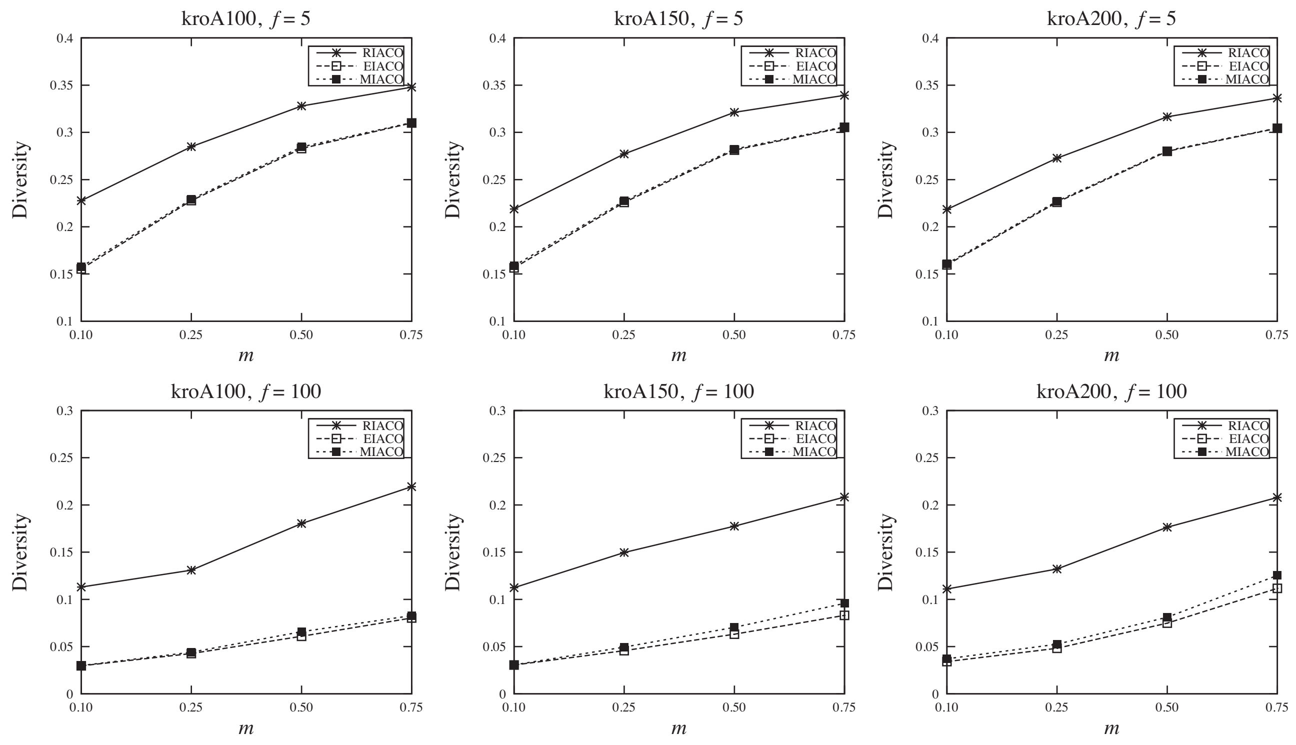

Fig. 4. Experimental results regarding the total diversity of the population of RIACO, EIACO and MIACO on random DTSPs.

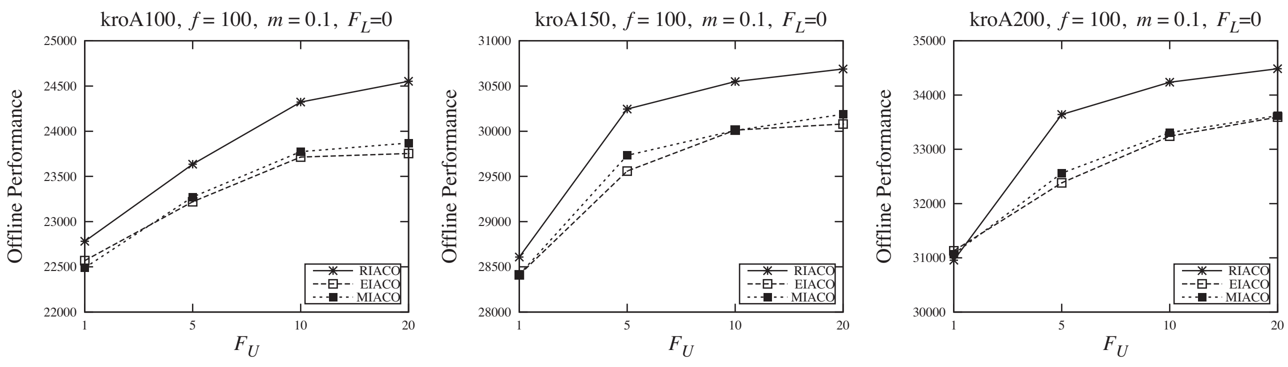

Fig. 5. Experimental results regarding the performance of ACO algorithms in cyclic DTSPs with different traffic bounds F_U.

Table 10
Experimental results regarding the offline performance on the DTSP with $m = rand[0, 1]$ and $f = rand[1, 100]$ in att532. "+" or "−" indicates that the algorithm in the row is significantly better than the one in the column or the opposite and "∼" indicates no significance.

Alg.	**RIACO**	**EIACO**	**MIACO**	$\mathcal{MM}$AS	P-ACO	SC-PACO	FS-PACO	M-ACO
Offline performance	147,646	145,968	146,240	150,436	156,287	152,848	150,185	155,176
Standard Deviation	523.14	507.91	585.91	292.57	691.61	756.33	643.04	503.27
				Wilcoxon Sum-Rank Test				
RIACO		−	−	+	+	+	+	+
EIACO	+		+	+	+	+	+	+
MIACO	+	−		+	+	+	+	+
$\mathcal{MM}$AS	−	−	−		+	+	∼	+
P-ACO	−	−	−	−		−	−	−
SC-PACO	−	−	−	−	+		−	+
FS-PACO	−	−	−	∼	+	+		+
M-ACO	−	−	−	−	+	−	−	

of the algorithms when $F_U = 1$, since MIACO performs better than EIACO whereas RIACO performs better than all its competitors in kroA200.

7.7. *Experimental study on comparing the proposed algorithms with other peer ACO algorithms*

In the basic experiments above, the performance of RIACO, EIACO and MIACO was investigated under two kinds of dynamic environments, i.e., random and cyclic, but with fixed f and m values. However, real world problems may involve different periods and severities of change. That is, each environment may change in different values of f and have different values of m on each environmental change. In order to investigate the performance of RIACO, EIACO and MIACO in such environments, further experiments were performed on a DTSP with varying f and m values, which were randomly generated with a uniform distribution in [1, 100] and [0, 1], respectively.

We compare the proposed algorithms with other peer ACO algorithms, i.e., $\mathcal{MM}$AS described in Section 4.2 and P-ACO, SC-PACO, FS-PACO and M-ACO described in Section 5. The parameter settings of the algorithms are described in Table 2. Note that 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 ($\tau - strategy$ and $\eta - strategy$) or the ACO-*shaking*, also described in Section 5, cannot be applied because they do not have any mechanism to detect dynamic changes.

The experimental results regarding the offline performance of the aforementioned algorithms are presented in Table 10 with the corresponding statistical test results. The dynamic behaviour of the algorithms and the m and f values used on each iteration are presented in Figs. 6 and 7, respectively. From the experimental results several observations can be drawn.

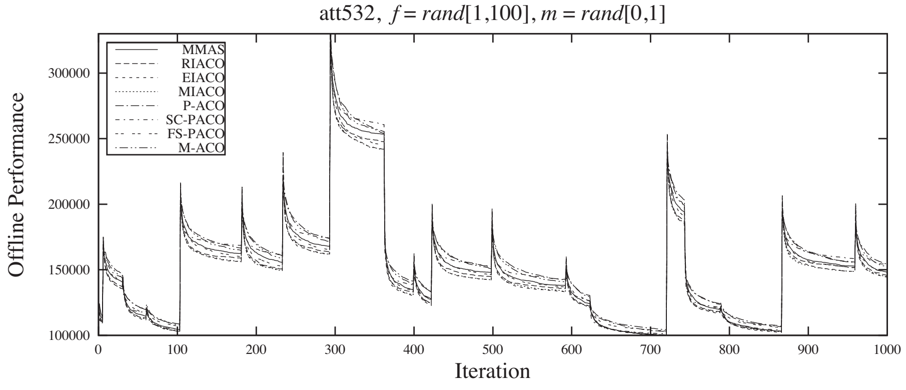

Fig. 6. Dynamic behaviour of ACO algorithms on the DTSP with $m = rand[0, 1]$ and $f = rand[1, 100]$.

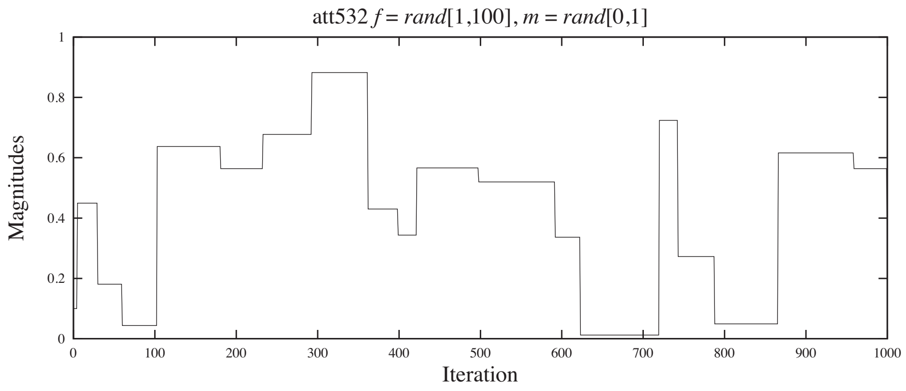

Fig. 7. The values of m and f used on the DTSP with $m = rand[0, 1]$ and $f = rand[1, 100]$.

First, among the proposed algorithms, EIACO performs significantly better than its competitors, followed by MIACO. This matches our previous experimental results where EIACO (and MIACO) perform better than RIACO in most dynamic cases.

Second, all the proposed algorithms perform significantly better than $\mathcal{MM}$AS and P-ACO. This is because $\mathcal{MM}$AS uses only pheromone evaporation to eliminate pheromone trails from the previous environment that are not useful to a new environment, and thus, needs sufficient time to adapt to the changing environments. As discussed previously $\mathcal{MM}$AS performs well when the environments are similar and enough time is available. For example, in the iterations between 620 and 720, where the value of m is not severe, $\mathcal{MM}$AS has a comparable performance; see Figs. 6 and 7. On the other hand, P-ACO eliminates pheromone trails directly if an ant is removed from $k_{long}(t)$. However, if identical ants are stored in the $k_{long}(t)$, then the algorithm will reach stagnation behaviour, and thus, needs sufficient time to escape from it.

Third, FS-PACO and SC-PACO performs significantly better than P-ACO, whereas the former algorithm performs significantly better than the latter algorithm. Both algorithms are variations of the P-ACO algorithm that aim to increase and maintain diversity in population-list. Hence, SC-PACO and FS-PACO adapt better to the changing environment than P-ACO probably because the simple crowding policy and the fitness sharing used in the algorithms, respectively, address the problem of identical ants in P-ACO.

Fourth, M-ACO is significantly better than P-ACO since the local search scheme that is integrated in M-ACO promotes exploitation to improve the solution quality, and the risk of stagnation behaviour is eliminated using a diversity increase scheme based on traditional immigrants. Whenever $k_{long}(t)$ contains identical ants, a random immigrant replaces an ant until the algorithm generates sufficient diversity.

8. Conclusions and future work

Different immigrants schemes have been successfully applied to EAs to address different DOPs [52,53]. In this paper, we integrate immigrants schemes with ACO to address a permutation-encoded DOP, i.e., the 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 re-appear. Three algorithms are proposed, i.e., RIACO, EIACO, and MIACO, where random immigrants, elitism-based immigrants, and memory-based immigrants are generated, respectively, and replace the worst ants in the current population in order to transfer knowledge to the pheromone trails and maintain diversity, which is important when addressing DOPs.

From the experimental studies of comparing the proposed algorithms on different cases of DTSPs, the following concluding remarks can be drawn. First, immigrants enhance the performance of ACO for DTSPs. Second, RIACO performs well in quickly and significantly changing environments. Third, EIACO performs well in slowly and slightly changing environments. Fourth, MIACO performs well in cyclic changing environments, where previous environments will re-appear. Fifth, the proposed algorithms perform better than other peer ACO algorithms in DTSPs. Sixth, the evaporation rate in $\mathcal{MM}$AS and the long-term memory size in MIACO depends on the magnitude of a dynamic change. Seventh, a high level of diversity or too much knowledge transferred do not always enhance the performance of ACO in DTSPs. Hence a good balance between the diversity generated and the knowledge transferred achieves good performance in DTSPs. Finally, a small immigrants replacement rate is usually a good choice for RIACO, EIACO and MIACO in DTSPs.

For future work, it will be interesting to adapt the replacement rate instead of using a small fixed value [61]. Another future work is to include the dynamic time-linkage (DTL) property [4,41], where the value at time t is dependent on at least one earlier solution, to the traffic factors. DTL is very common in real-world applications. For example, in a car navigation systems where traffic factors are considered, if the optimal routing path for a specific destination is provided to all cars at the current moment, then the traffic on the same “optimal” route will increase and make the route no longer optimal or congested in the future.

Acknowledgement

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of U.K. under grant EP/K001310/1.

References

[1] D. Angus, Niching for population-based ant colony optimization, in: Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing, 2006, pp. 15–22.
[2] D. Angus, T. Hendtlass, Dynamic ant colony optimisation, Applied Intelligence 23 (2005) 33–38.
[3] J.E. Bell, P.R. McMullen, Ant colony optimization techniques for the vehicle routing problem, Advanced Engineering Informatics 18 (2004) 41–48.
[4] P.A.N. Bosman, Learning, anticipation and time-deception in evolutionary online dynamic optimization, in: Proceedings of the 2005 Genetic and Evolutionary Computation Conference, ACM Press, 2005, pp. 39–47.
[5] B. Bullnheimer, R. Haïti, C. Strauss, An improved ant system algorithm for the vehicle routing problem, Annals of Operations Research 89 (1999) 319–328.
[6] B. Bullnheimer, R.F. Hartl, C. Strauss, A new rank based version of the ant system—a computational study, Central European Journal for Operations Research in Economics 7 (1) (1990) 25–38.

[7] E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, New York, 1999.
[8] J. Branke, Memory enhanced evolutionary algorithms for changing optimization problems, in: Proceedings of the 1999 IEEE Congress on Evolutionary Computation, vol. 3, IEEE Press, 1999, pp. 1875–1882.
[9] J. Branke, T. Kaußler, C. Schmidt, H. Schmeck, A multi-population approach to dynamic optimization problems, in: Proceedings of the 4th International Conference on Adaptive Computing in Design and Manufacturing, Springer-Verlag, 2000, pp. 299–308.
[10] J. Branke, E. Salihoğlu, C. Uyar, Towards an analysis of dynamic environments., in: Proceedings of the 2005 Genetic and Evolutionary Computation Conference, ACM Press, 2005, pp. 1433–1440.
[11] C. Cruz, J.R. González, D.A. Pelta, Optimization in dynamic environments: a survey on problems, methods and measures, Soft Computing. 15 (7) (2011) 1427–1448.
[12] H.G. Cobb, An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent non-stationary environments, Technical Report AIC-90-001, Naval Research Laboratory, Washington, USA, 1990.
[13] H.G. Cobb, J.J. Grefenstette, Genetic algorithms for tracking changing environments, in: Proceedings of the 5th International Conference on Genetic Algorithms, 1993, pp. 523–530.
[14] A. Colorni, M. Dorigo, V. Maniezzo, Distributed optimization by ant colonies, in: Proceedings of the 1st European Conference on Artificial Life, 1992, pp. 134–142.
[15] O. Cordón, I.F. de Viana, F. Herrera, L. Moreno, A new ACO model integrating evolutionary computation concepts: the best worst Ant System, in: Proceedings of the 2nd International Workshop on Ant Algorithms, Springer-Verlag, 2000, pp. 22–29.
[16] G. Di Caro, M. Dorigo, Ant Net: distributed stigmergetic control for communications networks, Journal of Artificial Intelligence Research 9 (1998) 317–365.
[17] G. Di Caro, F. Ducatelle, L.M. Gambardella, AntHocNet: an ant-based hybrid routing algorithm for mobile ad hoc networks, in: Proceedings of the 8th International Conference on Parallel Problem Solving from Nature, Springer-Verlag, 2004, pp. 461–470.
[18] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics 26 (1) (1996) 29–41.
[19] M. Dorigo, L.M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 53–66.
[20] M. Dorigo, T. Stützle, Ant Colony Optimization, The MIT Press, London, England, 2004.
[21] C.J. Eyckelhof, M. Snoek, Ant Systems for a Dynamic TSP, in: Proceedings of the 3rd International Workshop on Ant Algorithms, Springer-Verlag, 2002, pp. 88–99.
[22] M.R. Garey, D.S. Johnson, Computer and Intractability: A Guide to the Theory of NP-completeness, San Francisco, Freeman, 1979.
[23] L.M. Gambardella, E.D. Taillard, M. Dorigo, Ant colonies for the quadratic assignment problem, Journal of the Operations Research Society 50 (1999) 167–176.
[24] J.J. Grefenestette, Genetic algorithms for changing environments., in: Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature, 1992, pp. 137–144.
[25] M. Guntsch, M. Middendorf, Applying population based ACO to dynamic optimization problems, in: Proceedings of the 3rd International Workshop on Ant Algorithms, LNCS 2463, Springer-Verlag, 2002, pp. 111–122.
[26] M. Guntsch, M. Middendorf, A population based approach for ACO., in: EvoWorkshops 2002: Applications of Evolutionary Computing, Springer-Verlag, 2002, pp. 72–81.
[27] M. Guntsch, M. Middendorf, Pheromone modification strategies for ant algorithms applied to dynamic TSP., in: EvoWorkshops 2001: Applications of Evolutionary Computing, Springer-Verlag, 2001, pp. 213–222.
[28] M. Guntsch, M. Middendorf, H. Schmeck, An ant colony optimization approach to dynamic TSP, in: Proceedings of the 2001 Genetic and Evolutionary Computing Conference, 2001, pp. 860–867.
[29] T. Guo, Z. Michalewicz, Inver-over operator for the TSP, in: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, 1998, pp. 803–812.
[30] J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975.
[31] Y. Jin, J. Branke, Evolutionary optimization in uncertain environments—a survey, IEEE Transactions on Evolutionary Computation 9 (3) (2005) 303–317.
[32] P. Kilby, P. Prosser, P. Shaw, Dynamic VRPs: A study of scenarios, Technical Report APES-06-1998, University of Strathclyde, UK, 1998.
[33] S. Lin, B.W. Kerninghan, An effective heuristic algorithm for the traveling salesman problem, Operations Research 21 (2) (1973) 498–516.
[34] V. Maniezzo, A. Colorni, The ant system applied to the quadratic assignment problem, IEEE Transactions on Knowledge and Data Engineering 9 (5) (1999) 769–778.
[35] M. Mavrovouniotis, S. Yang, Ant colony optimization with immigrants schemes for dynamic environments, in: Proceedings of the 11th International Conference on Parallel Problem Solving from Nature, LNCS 6238, Springer-Verlag, 2010, pp. 371–380.
[36] M. Mavrovouniotis, S. Yang, A memetic ant colony optimization algorithm for the dynamic travelling salesman problem, Soft Computing 15 (7) (2011) 1405–1425, Springer-Verlag.
[37] M. Mavrovouniotis, S. Yang, Memory-based immigrants for ant colony optimization in changing environments, in: EvoApplications 2011: Applications of Evolutionary Computing, LNCS 6624, Springer-Verlag, 2011, pp. 324–333.
[38] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, third ed., Springer-Verlag, Berlin, 1999.
[39] R. Montemanni, L. Gambardella, A. Rizzoli, A. Donati, Ant colony system for a dynamic vehicle routing problem, Journal of Combinatorial Optimization. 10 (4) (2005) 327–343.
[40] T.T. Nguyen, Z. Yang, S. Bonsall, Dynamic time-linkage problems—the challenges, in: Proceedings of the International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future, IEEE Press, 2012, pp. 1–6.
[41] T.T. Nguyen, S. Yang, J. Branke, Evolutionary dynamic optimization: a survey of the state of the art, Swarm and Evolutionary Computation. 6 (2012) 1–24.
[42] A.E. Rizzoli, R. Montemanni, E. Lucibello, L.M. Gambardella, Ant colony optimization for real-world vehicle routing problems—from theory to applications, Swarm Intelligence 1 (2) (2007) 135–151.
[43] T. Stützle, H. Hoos, The MAX-MIN ant system and local search for the traveling salesman problem, in: Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, IEEE Press, 1997, pp. 309–314.
[44] T. Stützle, H. Hoos, MAX-MIN ant system, Future Generation Computer Systems 8 (16) (2000) 889–914.
[45] R.K. Ursem, Multinational GA optimization techniques in dynamic environments, in: Proceedings of the 2000 Genetic and Evolutionary Computation Conference, 2000, pp. 19–26.
[46] F. Vavak, K.A. Jukes, T.C. Fogarty, Performance of a genetic algorithm with variable local search range relative to frequency for the environmental changes, in: Proceedings of the 1998 International Conference on Genetic Programming, Morgan Kaufmann, 1998, pp. 22–25.
[47] H. Wang, D. Wang, S. Yang, A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems, Soft Computing. 13 (8–9) (2009) 763–780.
[48] M. Wineberg, F. Oppacher, Enhancing the GA's ability to cope with dynamic environments., in: Proceedings of the 2000 Genetic and Evolutionary Computation Conference, 2000, pp. 3–10.
[49] S. Yang, Memory-based immigrants for genetic algorithms in dynamic environments, in: Proceedings of the 2005 Genetic and Evolutionary Computation Conference, vol. 2, ACM Press, 2005, pp. 1115–1122.
[50] S. Yang, On the design of diploid genetic algorithms for problem optimization in dynamic environments, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation, IEEE Press, 2006, pp. 1362–1369.
[51] S. Yang, Genetic algorithms with elitism based immigrants for changing optimization problems, in: EvoWorkshops 2007: Applications of Evolutionary Computing, LNCS 4448, Springer-Verlag, 2007, pp. 627–636.
[52] S. Yang, Genetic algorithms with memory and elitism based immigrants in dynamic environments, Evolutionary Computation 16 (3) (2008) 385–416.
[53] S. Yang, H. Cheng, F. Wang, Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile ad hoc networks., IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews. 40 (1) (2010) 52–63.
[54] S. Yang, Y. Jiang, T.T. Nguyen, Metaheuristics for dynamic combinatorial optimization problems, IMA Journal of Management Mathematics (2012), Oxford University Press.
[55] S. Yang, R. Tinos, A hybrid immigrants scheme for genetic algorithms in dynamic environments, International Journal of Automation and Computing 4 (3) (2007) 243–254.
[56] S. Yang, Non-stationary problem optimization using the primal-dual genetic algorithm, in: Proceedings of the 2003 IEEE Congress on Evolutionary Computation, IEEE Press, 2003, pp. 2246–2253.
[57] S. Yang, X. Yao, Population-based incremental learning with associative memory for dynamic environments, IEEE Transactions on Evolutionary Computation 12 (5) (2008) 542–561.
[58] A. Younes, O. Basir, P.A. Calamai, Benchmark generator for dynamic optimization., in: Proceedings of the 3rd WSEAS International Conference on Soft Computing, Optimization, Simulation & Manufacturing Systems 3(1), 2004, pp. 273–278.
[59] E.L. Yu, P.N. Suganthan, Evolutionary programming with ensemble of explicit memories for dynamic optimization, in: Proceedings of the 2009 IEEE Congress on Evolutionary Computation, IEEE Press, 2009, pp. 431–438.
[60] X. Yu, K. Tang, X. Yao, An immigrants scheme based on environmental information for genetic algorithms in changing environments., in: Proceedings of the 2008 IEEE Congress of Evolutionary Computation, IEEE Press, 2008, pp. 1141–1147.
[61] X. Yu, K. Tang, X. Yao, Immigrant schemes for evolutionary algorithms in dynamic environments: Adapting the replacement rate, Science China Series F: Information Sciences 53 (1) (2010) 1–11.
[62] X. Yu, K. Tang, T. Chen, X. Yao, Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization, Memetic Computing. 1 (1) (2009) 3–24.