Soft Computing manuscript No.
(will be inserted by the editor)

A memetic ant colony optimization algorithm for the dynamic travelling salesman problem

Michalis Mavrovouniotis · Shengxiang Yang

Abstract Ant colony optimization (ACO) has been successfully applied for combinatorial optimization problems, e.g., the travelling salesman problem (TSP), under stationary environments. In this paper, we consider the dynamic TSP (DTSP), where cities are replaced by new ones during the execution of the algorithm. Under such environments, traditional ACO algorithms face a serious challenge: once they converge, they cannot adapt efficiently to environmental changes. To improve the performance of ACO on the DTSP, we investigate a hybridized ACO with local search (LS), called Memetic ACO (M-ACO) algorithm, which is based on the population-based ACO (P-ACO) framework and an adaptive inver-over operator, to solve the DTSP. Moreover, to address premature convergence, we introduce random immigrants to the population of M-ACO when identical ants are stored. The simulation experiments on a series of dynamic environments generated from a set of benchmark TSP instances show that LS is beneficial for ACO algorithms when applied on the DTSP, since it achieves better performance than other traditional ACO and P-ACO algorithms.

Keywords Memetic algorithm · Ant colony optimization · Dynamic optimization problem · Travelling salesman problem · Inver-over operator · Local search · Simple inversion · Adaptive inversion

Michalis Mavrovouniotis
Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
E-mail: mm251@mcs.le.ac.uk

Shengxiang Yang
Department of Information Systems and Computing
Brunel University, Uxbridge, Middlesex UB8 3PH, UK
E-mail: shengxiang.yang@brunel.ac.uk

1 Introduction

Ant colony optimization (ACO) algorithms have been successfully applied for different combinatorial optimization problems [14]. Traditionally, researchers have been focused on stationary optimization problems, where the environment remains fixed during the execution of an algorithm. However, many real-world applications are subject to dynamic environments. Such a problem becomes more challenging since the aim of an algorithm is not just to find the optimum of the problem, but to track the changing optima when changes occur [27].

Traditional ACO algorithms have been designed for stationary optimization problems [13], and may not be sufficient for dynamic optimization problems (DOPs). This is due to the fact that the pheromone trails of the previous environment will not make sense for the new environment after a change occurs. Furthermore, they cannot adapt well once the population converges into an optimum, since a certain level of diversity is vital to maintain the high quality of output efficiently. A simple way to address this problem is to re-initialize the pheromone trails and consider every change as the arrival of a new problem instance which needs to be solved from scratch. Unfortunately, this restart strategy is computationally expensive and usually not efficient.

The first application of an ACO algorithm for DOPs is the AntNET [11], which has been applied for the routing problem in communication networks. Other specialized pheromone strategies have been proposed for DOPs, which include local and global restart strategies [22], pheromone manipulation schemes to maintain or increase diversity [2, 15, 37], and memory-based approaches [20]. These methods have been applied to the dynamic travelling salesman problem (DTSP) due to its importance for many real-world applications. These strategies re-initialize the pheromone trails, as a restart strategy when a change occurs, but with guid-

ance in order to help the population to adapt efficiently to the new environment.

In recent years, there has been an increasing interest on a class of hybrid evolutionary algorithms (EAs) [30], called *memetic algorithms* (MAs), where local search (LS) operators are hybridized with EAs to improve the solution quality. MAs have been used for solving many optimization problems, such as arc routing problems [38,53,54,59], scheduling problems [19,36,60,61], and other applications [1,33, 43,65]. They have been applied for optimization problems under both stationary and dynamic environments. However, almost all existing MAs follow the framework of EAs and very rarely the framework of ACO. Thus, it would be interesting to investigate the performance and robustness of ACO-based MAs, which has been discussed that they are able to improve both measurements [9,34].

In this paper, a memetic ACO-based (M-ACO) algorithm, based on the population-based ACO (P-ACO) framework, is proposed and applied to the DTSP. In the M-ACO algorithm, we use multiple LS operators based on the inver-over (IO) method [24], which is a specialized operator for the TSP. This can enhance the solution quality of the algorithm, since it provides strong exploitation to the search process. On the other hand, it may not be effective for DOPs since we need to maintain a certain level of diversity within the population to adapt well in DOPs. Since immigrants schemes have been found beneficial when integrated with ACO for changing environments [37], we develop a diversity scheme based on random immigrants.

The rest of the paper is outlined as follows. Sect. 2 addresses the concept of DOPs and briefly reviews the application of EAs (including ACO) and MAs for DOPs. Sect. 3 describes the DTSP, which is the focus of this paper. Sect. 4 presents the standard ACO (S-ACO) and P-ACO algorithms, indicating their differences. Moreover, a description on how traditional S-ACO and P-ACO algorithms respond to dynamic changes is also given. Sect. 5 presents the proposed M-ACO algorithms for the DTSP, including the LS operators, with their adaptive mechanism, and the diversity scheme used. Sect. 6 presents the experimental results and analysis of the performance and robustness of the investigated algorithms. Finally, Sect. 7 gives the conclusions and discusses relevant future work.

2 Related work

2.1 EAs for DOPs

The environment in many real-world optimization problems, including the objective function, the decision variables, the problem instance, the constraints, etc., may change over time. As a result, the optimum of a problem may change as well. Optimization problems with such characteristics are known as DOPs [27].

One of the simplest ways to react to environmental changes is to consider each change as the arrival of a new problem and restart the algorithm to solve it from scratch [47]. However, this strategy requires substantial computational effort and time, while for DOPs usually the available re-optimization time is short.

Over the years, EAs have been a subject of an extensive research to solve DOPs since they can transfer knowledge from past environments. However, traditional EAs cannot adapt well to the new environment when changes occur once converged. To address the convergence problem, many strategies have been proposed to enhance the performance of EAs for DOPs, including diversity increasing or maintaining schemes [62,63], memory-based schemes [5], and multi-population schemes [6].

Similarly, ACO algorithms, which are a special class of EAs, suffer from exactly the same problem when applied for DOPs. Thus, inspired from the strategies for EAs, many ACO applications have also been developed for DOPs, such as P-ACO [20] which inherits EAs characteristics since it maintains a memory of limited size to store the best ants that are repaired when a dynamic change occurs and ACO with immigrants where new ants are introduced on every iteration (generation) into the population using different schemes, i.e., random immigrants, elitism-based immigrants, and hybrid immigrants, to increase the diversity [37].

2.2 MAs for DOPs

Apart from the strategies described above, which are used to address the convergence problem of EAs, MAs have also been used for DOPs [16–18,55,58]. However, existing MAs that are applied for either stationary or dynamic environments usually follow the framework of EAs [55], and rarely the ACO framework.

LS operators have also been found effective when applied with ACO [50]. Especially, on graph problems, e.g., the TSP, ACO algorithms have a great advantage over traditional EAs. The reason is due to the use of heuristic information, which is available from the problem instance itself, i.e., the distance between cities. Such information enables ACO algorithms to have prior knowledge of the problem from the initial stage, whereas EAs usually start at random.

It is important to mention that ACO algorithms have extremely bad results when such heuristic information is not used. Therefore, for problems where heuristic information is not available, EAs may be a more suitable choice than ACO algorithms. But, ACO becomes also suitable when it is applied with a LS operator. Experiments in [14, pp. 97–98] show that the best performing ACO algorithm, i.e., Max–

Min Ant System (MMAS) [51], without the use of heuristic information but with a LS operator, is able to generate good solutions. More specifically, experiments on the TSP showed that the MMAS with the well-known 2-opt operator [32] improves significantly the performance of the algorithm [50]. Similarly, the solution quality of pure EAs is also significantly improved using a LS operator [46].

Furthermore, ACO-based MAs have been found beneficial on other applications, such as path planning in sparse graphs [35], where the solutions found by the ants undergo local improvement from a modified version of the 2-opt operator. Similarly, the combination of the 2-opt operator with another specialized operator improves the solution quality for the vehicle routing problem [64]. Moreover, ACO has been used as a LS operator for an EA to help the search process escape from a local optimum on the multiple sequence alignment problem [31]. However, almost all ACO-based MAs have been applied for stationary problems, with very rare exceptions, such as an extension of the AntNET algorithm applied on mobile ad hoc networks [56].

3 The DTSP

The TSP is one of the most popular and well-studied *NP*-hard combinatorial optimization problems. 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 vertexes and E is a set of edges. The collection of cities is represented by the set V and the connection between them by the set E. In addition, the distances between cities are associated with C set, which are denoted as $C = (c_{ij})$, where c_{ij} is the distance between city i and city j.

A lot of algorithms, either exact algorithms or approximation algorithms (also known as heuristics), have been proposed to solve the stationary TSP [32,39]. 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 [41,29].

The TSP becomes more challenging and realistic if it is subject to a dynamic environment. For example, a salesman wants to distribute items sold on different cities starting from its home city and returning after he visited all the cities to its 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, after some cities are visited long traffic delays may affect the route, which increases the time planed. The salesman will need a new alternative route fast, in order to avoid long traffic delays and complete the task.

In our DTSP case, cities from the set V may be replaced by new ones over time. In this way, both cities and their links may change during the execution. For our experiments, the dynamic environment is generated by taking away half of the cities from the actual problem instance to construct a spare pool of cities. The frequency of change, i.e., f, denotes how often new cities from the spare pool replace ones in the current pool (i.e., the cities left in the actual problem instance). The degree of change, i.e., m, denotes how many cities from the spare pool replace ones in the current pool. Therefore, a DTSP instance is generated as follows: every f iterations, a percentage m of randomly chosen cities from the spare pool are exchanged with the same percentage of random ones from the current pool. In this way, the size of the problem instance remains the same through the whole run. The objective of the DTSP is not only to efficiently provide the global optimum solution, but also to efficiently track the changing optimum through different environments.

4 ACO for DOPs

4.1 The S-ACO approach

The S-ACO algorithm was first proposed and applied for the stationary TSP [10], which imitates the behaviour of real ants when they search for food from their nest to the food sources. Ants communicate using pheromone, which is a chemical substance produced by them and is applied to their trails. The more pheromone on a specific trail, the higher the possibility of that trail to be followed by ants. Using this scheme, ants indirectly communicate and cooperate to complete their food searching task as efficiently as possible.

A population of μ ants construct solutions based on pheromone trails and some heuristic information. On the initial stage, all trails are initialized with an equal amount of pheromones, 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 while being on city i, probabilistically, as follows:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \quad \text{if } j \in N_i^k, \tag{1}$$

where τ_{ij} is the existing pheromone trail between city i and city j, η_{ij} is the heuristic information available a priori, which is defined as $1/d_{ij}$, where d_{ij} is the distance between city i and city j. N_i^k denotes the neighbourhood of cities of ant k when being on city i. α and β are the two parameters that determine the relative influence of pheromone trail and heuristic information, respectively. With the probability q_0,

the ant k chooses the next city with the maximum probability, i.e., city z, which satisfies the following formula:

$$z = \underset{j \in N_i^k}{\operatorname{argmax}} \left[\tau_{ij}\right]^{\alpha} \left[\eta_{ij}\right]^{\beta} \quad (2)$$

This process continues until all ants have visited all cities once. Thereafter, ants deposit pheromone in order to update their pheromone trails. Ants retrace their solutions and deposit pheromone according to their solution quality on the corresponding trails, as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{k}, \forall\,(i,j) \in T^{k}, \quad (3)$$

where $\Delta\tau_{ij}^{k} = 1/C^{k}$ is the amount of pheromone that ant k deposits and C^{k} is the cost of the tour T^{k} constructed by ant k. In addition, a constant amount of pheromone is deduced from all trails due to the pheromone evaporation, which is defined as:

$$\tau_{ij} \leftarrow (1-\rho)\,\tau_{ij}, \forall\,(i,j), \quad (4)$$

where $0 < \rho \leq 1$ is the rate of evaporation. Lowering the pheromone values enables the algorithm to forget bad decisions made in previous iterations [14].

Several variations of the first ACO algorithm, i.e., the Ant System (AS) [10], have been proposed to improve its performance. They differ in the way pheromone trails are updated [4,12,50]. One of the most studied and best performing variation is the MMAS [51]. The main difference of MMAS from AS lies in the way that pheromone trails are manipulated. In MMAS, either the best-so-far ant or the iteration-best ant is allowed to update their pheromone trails. However, this behaviour may lead the algorithm into a stagnation behaviour, where all ants follow the same tour due to the high intensity of pheromone trails. To address this issue, the pheromone trail values 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 or when no improved tour has been found for several iterations [14].

Ants in MMAS construct their solutions as in AS and the evaporation of pheromone is also applied as in Eq. (4) but with a smaller evaporation rate. On the other hand, the deposit of new pheromone is defined as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{\text{best}}, \forall\,(i,j) \in T^{\text{best}}, \quad (5)$$

where T^{best} is the tour of the best ant and $\Delta\tau_{ij}^{\text{best}} = 1/C^{\text{best}}$, 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 $\Delta\tau_{ij}^{\text{best}} = 1/C^{bs}$, where C^{bs} is the tour cost of the best-so-far ant, or the iteration-best ant, in which case $\Delta\tau_{ij}^{\text{best}} = 1/C^{ib}$, where C^{ib} is the tour cost of the best ant of the current iteration.

Algorithm 1 Standard Ant Colony Optimization (S-ACO)

```
Initialize parameters α, β, q_0, ρ, and μ
Initialize pheromone trails τ_init
while termination condition not satisfied do
    for k := 1 to μ do
        Construct a solution by ant k
        Update statistics
    end for
    Evaporate pheromone with the rate ρ using Eq. 4
    if AS is selected then
        for k := 1 to μ do
            Deposit pheromone regarding ant k using Eq. 3
        end for
    end if
    if MMAS is selected then
        best := find the best ant
        Deposit pheromone regarding ant best using Eq. 5
        Limit pheromone trails in the range [τ_max, τ_min]
    end if
    if environmental change is detected then
        Re-initialize pheromone trails τ_init
    end if
end while
```

Algorithm 1 shows the framework of S-ACO algorithms with variants AS and MMAS. MMAS is much more exploratory than AS since its different pheromone manipulation scheme maintains a certain level of diversity, which avoids the population to get trapped in a local optimum solution.

4.2 The P-ACO approach

The P-ACO algorithm, shown in Algorithm 2, is a memory-based version of the S-ACO algorithm, which was first applied on the stationary TSP [21] and later on the DTSP [23]. It differs from the S-ACO framework since it maintains a memory (population-list) of limited size, which is used to store the best ants. The population-list is used every iteration to update the pheromone, instead of the whole population. The pheromone trails depend on the ants stored in the population-list. Evaporation of pheromone is not applied in P-ACO.

The initial phase and the first iterations of the P-ACO algorithm work in the same way as in the S-ACO algorithm. The pheromone trails are initialized with an equal amount of pheromone and the population-list M of size K is empty. For the first K iterations, the iteration-best ant deposits a constant amount of pheromone, which is defined as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{k}, \forall\,(i,j) \in T^{k}, \quad (6)$$

where $\Delta\tau_{ij}^{k} = (\tau_{max} - \tau_{init})/K$. Moreover, τ_{max} and τ_{init} denote the maximum and initial pheromone amount, respectively. This positive update procedure of Eq. 6 is performed whenever an ant enters the population-list.

Algorithm 2 Population-Based Ant Colony Optimization (P-ACO)

```
Initialize parameters α, β, q0, μ, and K
Initialize pheromone trails τ_init
M := empty
while termination condition not satisfied do
    for k := 1 to μ do
        Construct a solution by ant k
        Update statistics
    end for
    best := find the best ant
    M.enQueue(best)
    Add pheromone using Eq. 6
    if M is full then
        M.deQueue()
        Delete pheromone using Eq. 7
    end if
    if environmental change is detected then
        Re-initialize pheromone trails τ_init
        Repair ants in M
        for each ant k in M do
            Deposit pheromone regarding ant k using Eq. 6
        end for
    end if
end while
```

On iteration $K + 1$, the iteration-best ant enters the population-list and updates its pheromone trails positively. However, the ant that entered the population-list first needs to be removed in order to make room for the iteration-best ant, and thus, a negative constant update to its corresponding pheromone trails is done, which is defined as:

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

where $\Delta\tau_{ij}^{k}$ is defined as in Eq. 6. This mechanism keeps the pheromone trails between a certain τ_{min} value, which is equal to τ_{init}, and a τ_{max} value, which can be calculated by $\tau_{init} + \sum_{k=1}^{K} \Delta\tau_{ij}^{k}$.

This strategy is based on the *Age* of ants [20]. However, other strategies have also been proposed by researchers, such as *Quality* and *Prob* [20]. From the experimental results in [20], the default *Age* strategy is more consistent and performs better than the others, since other strategies have more chances to maintain identical ants into the population-list, which leads the algorithm to the stagnation behaviour. This is due to the fact that high levels of pheromone will be generated into a single trail and dominate the search space. Moreover, we have seen the importance of keeping the pheromone trails within certain bounds from the MMAS [50], which is one of the state-of-the-art ACO algorithms for stationary problems [2].

4.3 Response to environmental changes

Theoretically, ACO algorithms are able to adapt to dynamic changes since they are inspired from nature, which is a continuous changing process [27]. Technically, they can adapt by transferring knowledge from past environments [8]. So far, the description of ACO algorithms above has been made assuming stationary environments. Considering the DTSP, ACO needs to be modified in order to adapt to environmental changes efficiently.

The dynamics used in our DTSP test case, i.e., adding or deleting cities, affects the genotypic level and, usually, the phenotypic level of the ant. Therefore, considering that the solutions are affected by the change in iteration i, the pheromone trails will not make sense in iteration $i + 1$. For the S-ACO algorithms that follow the traditional framework, it is vital to re-initialize the pheromone trails to τ_{init} after a dynamic change, which acts as a restart of the algorithm.

For the P-ACO approach, the solutions stored in the population-list are repaired heuristically and the pheromone trails are re-generated accordingly. This strategy is called *KeepElitst* [23] and uses two greedy heuristics to repair the genotype of the population: (1) the cities that are no longer present are removed from the solutions; and (2) the new cities are placed individually in a greedy fashion where they cause the minimum increase on the phenotype.

5 Proposed M-ACO algorithms for the DTSP

5.1 Framework of ACO-based MAs

Our proposed M-ACO algorithm is not based on the S-ACO framework but on the P-ACO algorithm since P-ACO is more suitable for DOPs. Experiments show that P-ACO is competitive with a S-ACO with simple re-initialization of pheromones on even large and slow environmental changes, where the restart strategy has enough time to converge to a good optimum and the environments are not similar, which makes any knowledge transfer inefficient. Moreover, P-ACO is significantly better on slight and fast environmental changes [14, pp. 264–265].

In M-ACO, after constructing solutions, the best ant is selected to be improved by a LS operator. The LS operator used is an adaptive version of the IO operator, which is one of the leading operators for the TSP [24]. If a better solution is found after each LS step, it replaces the selected best ant. After the LS steps finish, the resulting ant is added into the population-list and the pheromone trails are updated accordingly, following the update policy of the P-ACO algorithm.

Furthermore, a diversity scheme based on random immigrants is applied with the M-ACO algorithm in case the ants within the population-list are identical. The traditional P-ACO algorithm faces a high risk to maintain identical ants into the population-list and prevents ants to explore other areas in the search space because a high intensity of pheromone is generated into a single trail. The diversity scheme checks whether the population-list keeps a certain diversity. If not, the random immigrants scheme is ac-

Algorithm 3 Proposed M-ACORI Algorithm

```
1: Initialize parameters α, β, q_0, μ, and K
2: Initialize pheromone trails τ_init
3: M := empty
4: while termination condition not satisfied do
5:    for k := 1 to μ do
6:       Construct a solution by ant k
7:       Update statistics
8:    end for
9:    best := find the best ant
10:   AIO(best) using Algorithm 5
11:   M.enQueue(best)
12:   Add pheromone using Eq. 6
13:   if M is full then
14:      M.deQueue()
15:      Delete pheromone using Eq. 7
16:   end if
17:   if diversity of M is zero then
18:      temp := generate a random immigrant
19:      Replace a randomly selected ant in M by temp
20:   end if
21:   if environmental change is detected then
22:      Re-initialize pheromone trails τ_init
23:      Repair ants in M
24:      for each ant k in M do
25:         Deposit pheromone regarding ant k using Eq. 6
26:      end for
27:   end if
28: end while
```

Algorithm 4 IO(S, p)

```
1: S' := S
2: select randomly a city c from S'
3: if rand() ≤ p then
4:    select the city c' from the remaining cities in S'
5: else
6:    select randomly another individual S'' from the population
7:    assign to c' the next city to the city c ∈ S''
8: end if
9: inverse the tour from the next city of city c to city c' in S'
10: if S' is better than S then
11:    S := S'
12: end if
```

tivated to replace one ant in the population-list with a random one, in order to help ants adapt well to the next environment. The complete framework of our proposed M-ACO algorithm with the random immigrants scheme, denoted M-ACORI, is shown in Algorithm 3.

5.2 Inver-over operator

It has been shown that the combination of LS heuristics and evolutionary systems enhances the performance of EAs and ACO algorithms on hard optimization problems [45, 50]. LS helps the population to escape local optimum efficiently and directs individuals (or ants) into promising areas in the search space.

The IO operator is one of the leading operators used as local optimizers in EAs for the stationary TSP [24]. It is based on the inversion operator, where two selected cities form a segment which is reversed. It has been applied with a S-ACO algorithm for the stationary TSP [3], where IO is performed among the best and second best ants, rather than the whole population, resulting in two offspring. The best two individuals from the pool created by the two parents and two offspring are selected. The LS operator is applied every iteration to help the ACO algorithm escape from local optimum, improve its solution quality, and increase its convergence speed. The algorithm was tested on relatively small stationary TSP instances (up to 75 cities) and shows a better global search ability and a higher convergence rate than a S-ACO algorithm and a hybrid ACO algorithm embedded with traditional crossover and mutation operators. Furthermore, it is shown that a large number of iterations of the IO operator does not lead to better results.

In fact, the IO operator can be seen as a new EA itself since it consists of crossover and mutation. The difference from a pure EA lies in that: (1) each individual competes with its offspring only, and (2) it uses only one operator, which is the combination of a blind inversion (BI) and a guided inversion (GI). The pseudo-code of IO is presented in Algorithm 4. The BI can be seen as a mutation operator and has a lower probability to be executed since the second city, which determines the size of the segment to be reversed, is selected randomly from the same individual of the first city. On the other hand, the GI can be seen as a crossover operator, where the second city is determined according to another individual randomly selected from the current population. An example of BI and GI on a TSP solution is presented in Fig. 1. The selection probability of BI and GI is determined by the parameter p in Line 3 of Algorithm 4. If $p = 1$, the inversion becomes a BI, and if $p = 0$, it becomes a GI. The parameter p used in the IO operator is set to 0.02 by default.

Both inversions provide exploration and maintain sufficient diversity within the population. However, once the population converges to a solution, the GI may not be effective anymore since the randomly selected individual has a high probability to be identical with the current one. Such behaviour may not be sufficient for the DTSP because no valid inversion will be performed, which decreases the exploration ability. However, it may be advantageous to the convergence speed of the population because the exploration provided is guided and can help to provide a solution efficiently in the DTSP. On the other hand, BI may delay the convergence speed, but can help the population to escape from a local optimum if it has converged to one. Therefore, both types of inversions can be useful for the DTSP.

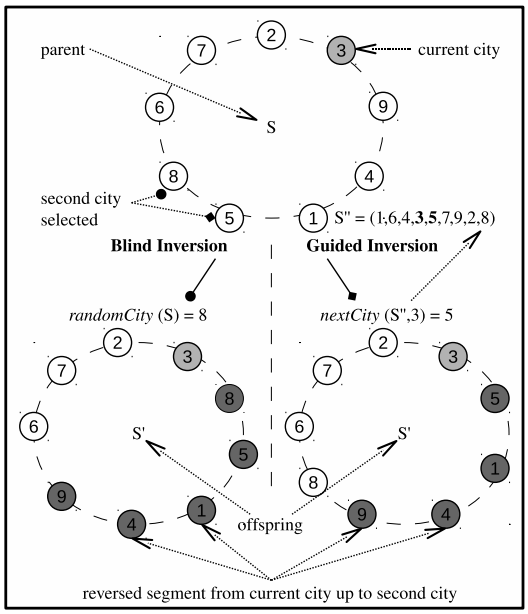

Fig. 1 Example of BI and GI operators on a TSP solution

5.3 Adaptive inver-over operator

As discussed above, the GI operator may be advantageous during the first iterations of the algorithm to enhance the convergence speed and the BI operator may be advantageous to enhance the solution quality. GI is not effective when the population has converged since, possibly all solutions within the population are identical, thus it cannot explore any new areas in the search space. During that period of time, BI may be effective to increase the diversity and move the population into new promising areas.

The use of multiple LS operators within a MA framework has been found advantageous to address the issue of problem-dependency of LS operators [49,52,55], e.g., each inversion type may be effective for some classes of problem instances. Also, it can be advantageous in different population evolution periods, e.g., BI is effective on later stages of the algorithm and GI is effective on the initial stages of the algorithm.

There are different adaptive mechanisms used to promote cooperation and competition between different LS operators. Multiple LS schemes can be executed in parallel based on competition and a learning mechanism can be used to give greater chances to those efficient LS operators to be used at a later stage [28,44]. On the other hand, multiple LS schemes can also be executed based on cooperation, where different LS operators are activated during different population evolution periods [9,42]. In the adaptive IO (AIO) operator that is applied in our proposed M-ACO algorithms, both BI and GI work together. The two inversion operators compete and cooperate in order to obtain the advantages of both

Algorithm 5 AIO($best$)

```
1: if an environmental change occurs then
2:     p_bi := p_gi := 0.5;
3: end if
4: ξ_bi := ξ_gi := 0
5: for i := 1 to ls do
6:     if rand() ≤ p_bi then
7:         IO(best, 1.0) using Algorithm 4
8:         Update ξ_bi
9:     else
10:        IO(best, 0.0) using Algorithm 4
11:        Update ξ_gi
12:    end if
13: end for
14: Re-calculate p_bi and p_gi
```

of them during different periods when they are effective, as described above.

BI and GI are denoted as an IO operator with $p = 1.0$ and $p = 0.0$, in Lines 7 and 10 of Algorithm 5, respectively. In the AIO operator, both BI and GI are selected probabilistically at every step of an LS operation on every iteration of the algorithm. Let p_{bi} and p_{gi} denote the probability of applying BI and GI to the individual selected for LS, respectively, where $p_{bi} + p_{gi} = 1$. Initially, the probabilities are both set to 0.5 in order to promote a fair competition between the two operators. The probabilities are adjusted according to the improvement each inversion operator has achieved on every LS step. The probability of the operator with the higher improvement will be increased using a similar mechanism as introduced in [55].

Let ξ denote the degree of improvement of the selected ant after an LS step, which is calculated as follows:

$$\xi = \frac{\left| C^{best'} - C^{best} \right|}{C^{best}}, \tag{8}$$

where $C^{best'}$ is the tour cost of the best ant after applying an LS step (using BI or GI) and C^{best} is the tour cost of the best ant before applying the LS step. When the number of LS steps reaches the preset step size, denoted as ls, the degree of improvement regarding BI and GI operators, denoted as ξ_{bi} and ξ_{gi}, respectively, is calculated and used to adjust the probabilities of selecting BI and GI in the next iteration, $p_{bi}(t+1)$ and $p_{gi}(t+1)$, as follows:

$$p_{bi}(t+1) = p_{bi}(t) + \xi_{bi}(t), \tag{9}$$

$$p_{gi}(t+1) = p_{gi}(t) + \xi_{gi}(t), \tag{10}$$

$$p_{bi}(t+1) = \frac{p_{bi}(t+1)}{p_{bi}(t+1) + p_{gi}(t+1)}, \tag{11}$$

$$p_{gi}(t+1) = 1 - p_{bi}(t+1), \tag{12}$$

where $\xi_{bi}(t)$ and $\xi_{gi}(t)$ are the total degree of improvement achieved by BI and GI operators at iteration t, respectively.

Both p_{bi} and p_{gi} are set to their initial value, i.e., 0.5, when an environmental change occurs in order to re-start the cooperation and competition when a new environment arrives. The pseudo-code of AIO is given in Algorithm 5.

5.4 Increasing and maintaining the population diversity

The main problem of EAs when applied on DOPs is their premature convergence. Once the algorithm has converged to an optimum, it cannot adapt well to the new environment when a change occurs. Even if LS is applied within an EA, the problem will still remain unsolved, or become even worse, because of the strong exploitation LS provides. As a result, the population will later on lose the capability of exploring new promising areas in the search space.

This problem becomes even bigger with ACO algorithms because of the way ants construct their solutions. They use heuristic information, i.e., the distance between cities for the TSP, and the population of ants is more likely to be identical from the first iterations. A natural way to address this problem is to increase the constant parameter of pheromone in Eq. (1), in order to force ants to consider more the pheromone trail values over the heuristic information values to increase exploration, when they construct their tours. But, then the algorithm faces a high risk of randomization.

Immigrants schemes have been found effective when applied in P-ACO for DTSPs [37], since they maintain a certain level of diversity within the population. The general idea of immigrants schemes is to introduce random immigrants, to replace a percentage of individuals in the current population every iteration [25]. However, in M-ACO there is no need to generate immigrants every iteration to let the LS work without disturbing its optimization process. Therefore, to both address the convergence issue and limit the disruption of the optimization process, random immigrants are introduced when a predefined diversity threshold is reached. This threshold is calculated by the diversity within the population-list, since it is based on the P-ACO algorithm. On every iteration i of run j, the diversity of the population-list is calculated as follows:

$$Div_{ij} = \frac{1}{K(K-1)} \sum_{p=1}^{K} \sum_{q \neq p}^{K} CE(p,q), \tag{13}$$

where K is the size of the population-list and $CE(p,q)$ is a similarity metric between ant p and ant q, which is calculated as follows:

$$CE(p,q) = 1 - \frac{\text{common edges}}{\text{number of cities}}. \tag{14}$$

If the diversity of the population-list, calculated by Eq. 13, is 0.0, it means that all the ants are identical. As a result, a high intensity of pheromone will be generated into one trail, forcing the population to converge into one path only. Therefore, every time the population reaches $Div = 0.0$, a random immigrant ant is generated to replace one current ant in the population-list.

6 Experimental study

In order to investigate the performance of proposed M-ACO algorithms for the DTSP, five sets of experiments are carried out in this study based on a set of test DTSPs constructed from benchmark stationary TSP instances. The first set of experiments aims to analyse the effect of LS operators on the performance of M-ACO algorithms for the stationary TSP instances. The second set of experiments analyses the effect of the diversity scheme (i.e., the random immigrants scheme) on the performance of M-ACO for the DTSP. The third set of experiments compares the overall performance of M-ACORI with other ACO algorithms from the literature for the DTSP. The fourth set of experiments compares the overall robustness of M-ACORI with other ACO algorithms from the literature for the DTSP. Finally, the fifth set of experiments investigates the performance of M-ACORI under dynamic environments of varying degrees of change speed and change severity.

6.1 Experimental setup

In the experiments, all the algorithms were tested on the DTSP instances that are constructed from three stationary benchmark TSP instances taken from TSPLIB,[1] which are visually illustrated in Fig. 2. Our implementation closely follows the guidelines of the ACOTSP[2] implementation, which is published by the ACO authors. By taking each stationary problem instance, we have generated dynamic environments using the method described in Sect. 3. The parameters f and m indicate the frequency and magnitude of dynamic changes. The f parameter is defined as the number of iterations between two environmental changes, and the m parameter is defined as the percentage of selected cities from the spare pool that replace other cities from the actual instance. The value of f was set to 20 and 100, indicating environmental changes of high and low frequencies, respectively. The value of m was set to 10, 25, 50, and 75, indicating the degree of environmental changes from small, to medium, to large, respectively. As a result, eight dynamic environments, i.e., two values of f $\times$ four values of m, were generated from each stationary TSP instance in order to systematically analyse the adaption and searching capabilities of each algorithm on the DTSP.

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

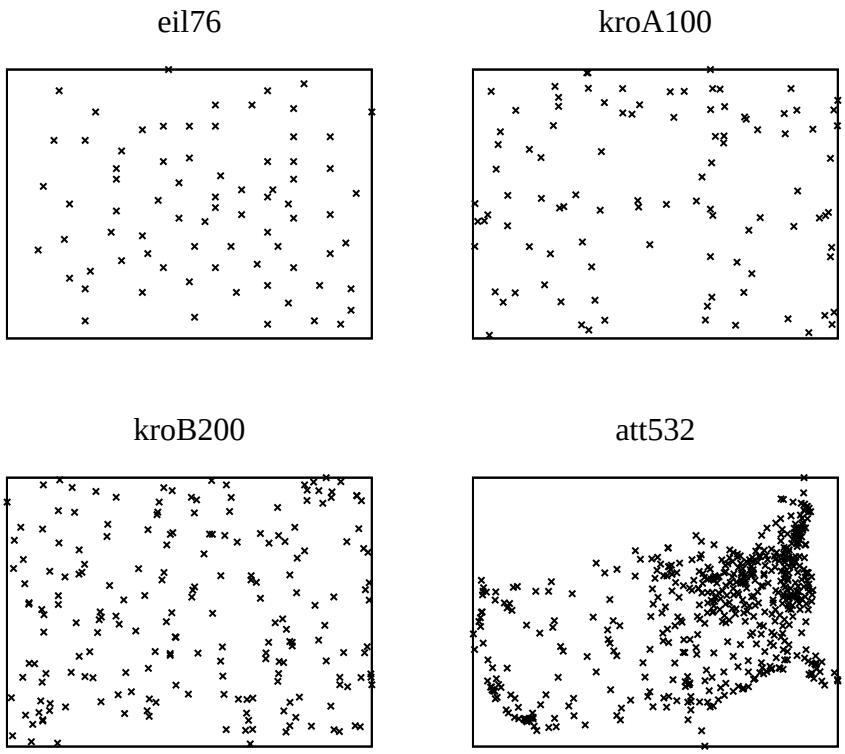

Fig. 2 The structure of the TSP instances used in the experiments.

Table 1 Parameter settings for the algorithms investigated, where C^{nn} is the length of a tour generated by the nearest-neighbour heuristic and c is the number of cities of the given problem instance

Algorithm	τ_{init}	α	β	ρ	K	q_0	μ	ls
M-ACOs	$1/(c-1)$	1	5	-	3	0.9	30	20
P-ACO	$1/(c-1)$	1	5	-	3	0.9	50	-
S-ACO	$1/\rho C^{nn}$	1	5	0.02	-	0.0	50	-

For each algorithm on a DTSP instance, $N = 30$ independent runs were executed on the same random environmental changes. The algorithms were executed for $G = 1000$ iterations (or generations) and an observation was taken on each iteration. The rest of the parameters used, and found beneficial, for the algorithms are shown in Table 1, where τ_{init} indicates the value of pheromone set in the initial phase, α and β determine the relative influence of pheromone and heuristic information on the construction of tours, respectively, ρ is the pheromone evaporation rate used in the S-ACO, but not for P-ACO and M-ACOs,[3] K is the length of the population-list for the P-ACO and M-ACOs, q_0 determines the greediness of the ant's decision rule and μ is the number of ants which determine the population size. The proposed M-ACO algorithm has a smaller population size than the others since they perform more evaluations with the LS steps, i.e., $ls = 20$. This is because we want to execute the same number of evaluations in each iteration for all algorithms in order to have a fair comparison. Note that S-ACO uses the MMAS framework, and P-ACO uses the *Age* update policy.

[3] The term M-ACOs denotes all ACO-based MAs used in the experiments

6.2 Measurements for dynamic environments

There are several performance measures that have been used to compare EAs (including ACO algorithms) for DOPs [40, 57]. However, there is not any agreed method to measure the behaviour [40], since researchers pay attention to different aspects when they analyze algorithms in dynamic environments [48]. Some researchers are interested in the extreme behaviour of the systems, i.e., the best result an algorithm can obtain, to investigate how well their system can do. On the other hand, others are interested in the population as a whole, i.e., the population distribution, to investigate and understand population-based algorithms as representation of evolutionary systems. In this paper, we use the measurements discussed in [48], i.e., *performance* and *robustness*, and both their best and average measures are examined and reported.

The performance is one of the most traditional measures of an algorithm which shows how well a system can do. The fitness of the best-so-far ant after a change is used to evaluate the performance. Therefore, the overall *Best Performance* P^{best} and overall *Average Performance* P^{avg} are defined, respectively, as follows:

$$P^{\text{best}} = \frac{1}{G}\sum_{i=1}^{G}(\frac{1}{N}\sum_{j=1}^{N}P_{ij}^{\text{best}}), \tag{15}$$

$$P^{\text{avg}} = \frac{1}{G}\sum_{i=1}^{G}(\frac{1}{N}\sum_{j=1}^{N}P_{ij}^{\text{avg}}), \tag{16}$$

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

The robustness is a measure which shows the persistence of the algorithm. The previous performance measure may not be enough to have a complete analysis of the algorithms for DOPs. Therefore, it is also important to measure the robustness of the algorithms [26]. As a result, the *Best Robustness* R_i^{best} and the *Average Robustness* R_i^{AVG} of iteration i over all runs, are defined, respectively, as follows:

$$R_i^{\text{best}} = \frac{1}{N}\sum_{j=1}^{N}R_{ij}^{\text{best}}, \tag{17}$$

$$R_i^{\text{avg}} = \frac{1}{N}\sum_{j=1}^{N}R_{ij}^{\text{avg}}, \tag{18}$$

where R_{ij}^{best} and R_{ij}^{avg} are the best robustness and the average robustness of iteration i of run j, respectively, which are defined as follows:

$$R_{ij}^{\text{best}} = \begin{cases} 1, & \text{if } \frac{P_i^{\text{best}}}{P_{i+1}^{\text{best}}} > 1 \\ \frac{P_i^{\text{best}}}{P_{i+1}^{\text{best}}}, & \text{otherwise,} \end{cases} \tag{19}$$

$$R_{ij}^{\text{avg}} = \begin{cases} 1, & \text{if } \frac{P_i^{\text{avg}}}{P_{i+1}^{\text{avg}}} > 1 \\ \frac{P_i^{\text{avg}}}{P_{i+1}^{\text{avg}}}, & \text{otherwise,} \end{cases} \tag{20}$$

A higher result from Eqs. 17 or 18 indicates a higher robustness level, which means that the solution quality is more persistent.

6.3 Experimental analysis on the effect of LS operators

We first study the effect of BI, GI and AIO LS operators when applied to P-ACO on the stationary test problems. Moreover, the diversity scheme of random immigrants is applied to enhance the performance of the MAs. We use M-ACORI1, M-ACORI2, and M-ACORI3 to denote M-ACORI with BI, M-ACORI with GI, and M-ACORI with the proposed AIO which combines both BI and GI, respectively. The experimental results are shown in Fig. 3 and several results can be observed.

First, it is obvious that all MAs outperform the P-ACO algorithm on all the test problems as expected since they converge to a much better solution. However, different MAs work better on different instances. M-ACORI1 locates a better solution on `kroA100`, M-ACORI2 on `eil76`, and M-ACORI3 on `kroB200`. This result indicates that LS operators are problem-dependent, which is natural since it is almost impossible to develop an algorithm to outperform the remaining algorithms on all problem instances.

Second, M-ACORI3, which uses the proposed AIO, always performs reasonably well since it combines both merits of BI and GI, i.e., the convergence speed and solution quality. M-ACORI3 converges equally fast as the other two algorithms on the first iterations. However, it requires slightly more time than the others to locate a good solution. Thus, it is a good choice for the DTSP and will be used for the rest of our experiments, and it will be denoted as M-ACORI for the rest of the paper.

6.4 Experimental analysis on the effect of the diversity scheme

In the second set of experiments, we study whether the diversity provided by the random immigrants scheme is effective on ACO-based MAs. In the experiments, we compare M-ACO and M-ACORI, which denotes an ACO-based MA without the diversity scheme and an ACO-based MA with the proposed diversity scheme, respectively. In the first section of Table 2, the results of the *Best Performance* are given and in the second section the corresponding statistical results of comparing the algorithms by the two-tailed t-test with 58 degree of freedom at a 0.05 level of significance are given. In Table 2, "$s+$" or "$s-$" indicates that the first algorithm or the second algorithm is significantly better, respectively, whereas "$\sim$" indicates that the two algorithms are statistically equivalent. From Table 2, several results can be observed.

First, the proposed diversity scheme efficiently improves the performance of M-ACO, since M-ACORI is significantly better in almost all dynamic test cases. This is due to the fact that once the population converges to an optimum, random immigrant ants are generated and replace other ants

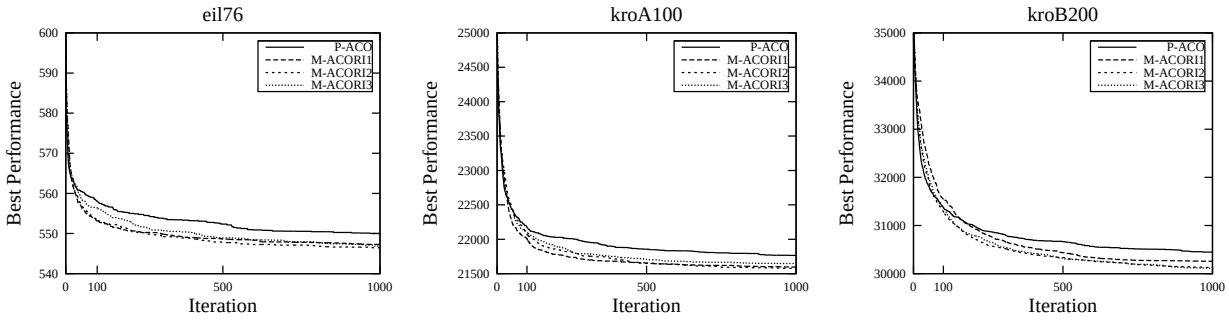

Fig. 3 Experimental results regarding the *Best Performance* of M-ACOs in comparison with traditional P-ACO on stationary test problems.

Table 2 Experimental results regarding the *Best Performance* of ACO-based MAs with diversity scheme (M-ACORI) and without a diversity scheme (M-ACO).

Algorithms	eil76				kroA100				kroB200			
Best Performance												
$f = 20$, $m \Rightarrow$	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
M-ACO	357.6	369.3	375.4	376.9	16568.4	16298.1	16634.4	16661.2	23004.4	23429.9	23174.2	23387.3
M-ACORI	356.9	368.7	374.6	375.9	16530.4	16269.6	16598.4	16631.3	22973.0	23392.5	23154.8	23391.4
$f = 100$, $m \Rightarrow$	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
M-ACO	363.1	367.3	364.3	374.8	16801.8	16177.3	16278.1	16391.4	22272.9	22913.6	22445.9	22796.2
M-ACORI	361.9	366.0	362.7	372.9	16724.9	16124.8	16210.3	16285.8	22174.0	22802.8	22319.8	22687.6
						t-Test Result						
$f = 20$, $m \Rightarrow$	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
M-ACO⇔M-ACORI	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$\sim$	$s-$	$\sim$	$\sim$
$f = 100$, $m \Rightarrow$	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
M-ACO⇔M-ACORI	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$

into the population-list and helps the LS to escape from a possible local optimum.

Second, on all test cases with $f = 100$, which denotes slow changing environments, M-ACORI is significantly better. However, on some cases with $f = 20$, which denotes fast changing environments, M-ACORI performs equivalent with M-ACO. The reason lies in the time the proposed AIO needs in order to be effective, as discussed in Sect. 6.3. Possibly a scheme that can provide more diversity may be suitable on cases where the environment changes fast.

Moreover, in Fig. 4, the *Total Diversity* of the two algorithms is presented, which is calculated as follows:

$$Div^{total} = \frac{1}{G}\sum_{i=1}^{G}(\frac{1}{N}\sum_{j=1}^{N} Div_{ij}), \tag{21}$$

where G is the total number of iterations, N is the total number of runs, and Div_{ij} is calculated as in Eq. 13 but for the actual ant population instead for the population-list M. The diversity of ACO, in general, can never reach a very high level. This is due to the guidance ants gain from the heuristic information, i.e, the distance between cities, when they are constructing their solutions. Therefore, even a small increase in the diversity of the population has a significant impact on the performance of the ACO algorithms. This can be observed from both Fig. 4 and Table 2, where M-ACORI maintains a higher diversity level than M-ACO and achieves better *Best Performance*.

6.5 Experimental study regarding the overall performance

In the third set of experiments, we study the overall performance (both *Best Performance* and *Average Performance*) of M-ACORI in comparison with other ACO algorithms (S-ACO and P-ACO). The comparison results are given in Table 3, by the two-tailed t-test with 58 degree of freedom at a 0.05 level of significance, where "$s+$", "$s-$", and "$\sim$" have the same meaning as in Table 2. In addition, to visually observe the dynamic performance of the algorithms, the results regarding the *Best Performance* and the *Average Performance* are plotted in Figs. 5 and 6, respectively. For both figures, the two values of m, i.e., $m = 10\%$ and $m = 75\%$, are plotted, which represent slight and significant changes, respectively. The f value with a fast frequency of change, i.e., $f = 20$, is used in Fig. 5 for the *Best Performance*, and with a slow frequency of change, i.e., $f = 100$, is used in Fig. 6 for the *Average Performance*. From Table 3 and Figs. 5 and 6, several results can be observed.

First, regarding the overall *Best Performance* results, the proposed M-ACORI algorithm significantly outperforms both P-ACO and S-ACO algorithms on almost all problem instances, with either slight or significant changes and with

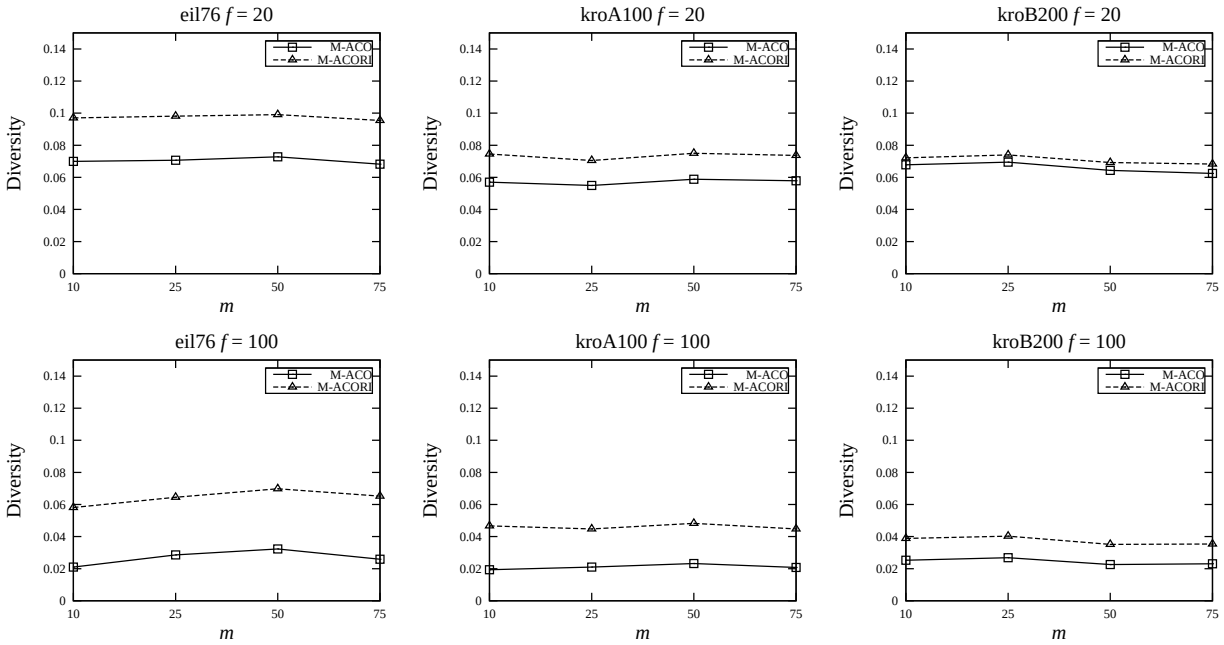

Fig. 4 Experimental results regarding the *Total Diversity* of population of M-ACORI in comparison with M-ACO on dynamic test problems.

Table 3 Statistical test results regarding the overall performance of comparing M-ACORI with other ACO algorithms.

Algorithms	eil76				kroA100				kroB200			
Best Performance												
$f = 20, m \Rightarrow$	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
S-ACO⇔P-ACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
S-ACO⇔M-ACORI	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
P-ACO⇔M-ACORI	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
$f = 100, m \Rightarrow$	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
S-ACO⇔P-ACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$-$	$s-$	$s-$	$s-$	$s-$	$s-$
S-ACO⇔M-ACORI	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
P-ACO⇔M-ACORI	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
Average Performance												
$f = 20, m \Rightarrow$	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
S-ACO⇔P-ACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
S-ACO⇔M-ACORI	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
P-ACO⇔M-ACORI	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
$f = 100, m \Rightarrow$	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
S-ACO⇔P-ACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$-$	$s-$	$s-$	$s-$	$s-$	$s-$
S-ACO⇔M-ACORI	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
P-ACO⇔M-ACORI	$\sim$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$

either fast or slow changes. The reason for this result is that the AIO operator, which is used as the LS operator in M-ACO, helps the population to converge fast at the early iterations, using GI, and later on maintains diversity with BI and the random immigrants scheme. This effect can be observed from Table 3 and Fig. 5, where the performance of M-ACORI is significantly better than P-ACO after a dynamic change. This is because P-ACO has a high risk of storing identical ants to the population-list, which leads to a high intensity of pheromone trails. The trails that contain a high intensity of pheromone lead the algorithm to a stagnation behaviour, i.e., the premature convergence of the P-ACO algorithm. However, the AIO operator with the diversity scheme used in M-ACORI is able to overcome this problem because it maintains a certain level of diversity in the population-list. As a result, the trails with a high intensity of pheromone are eliminated using the adaptive and evolutionary components of M-ACORI.

Second, it was expected that the S-ACO with a simple pheromone re-initialization strategy would outperform both P-ACO and M-ACORI algorithms on problem instances with significant and slowly changing environments, since the ants stored in the population-list of P-ACO and M-ACORI will not make much sense when different environ-

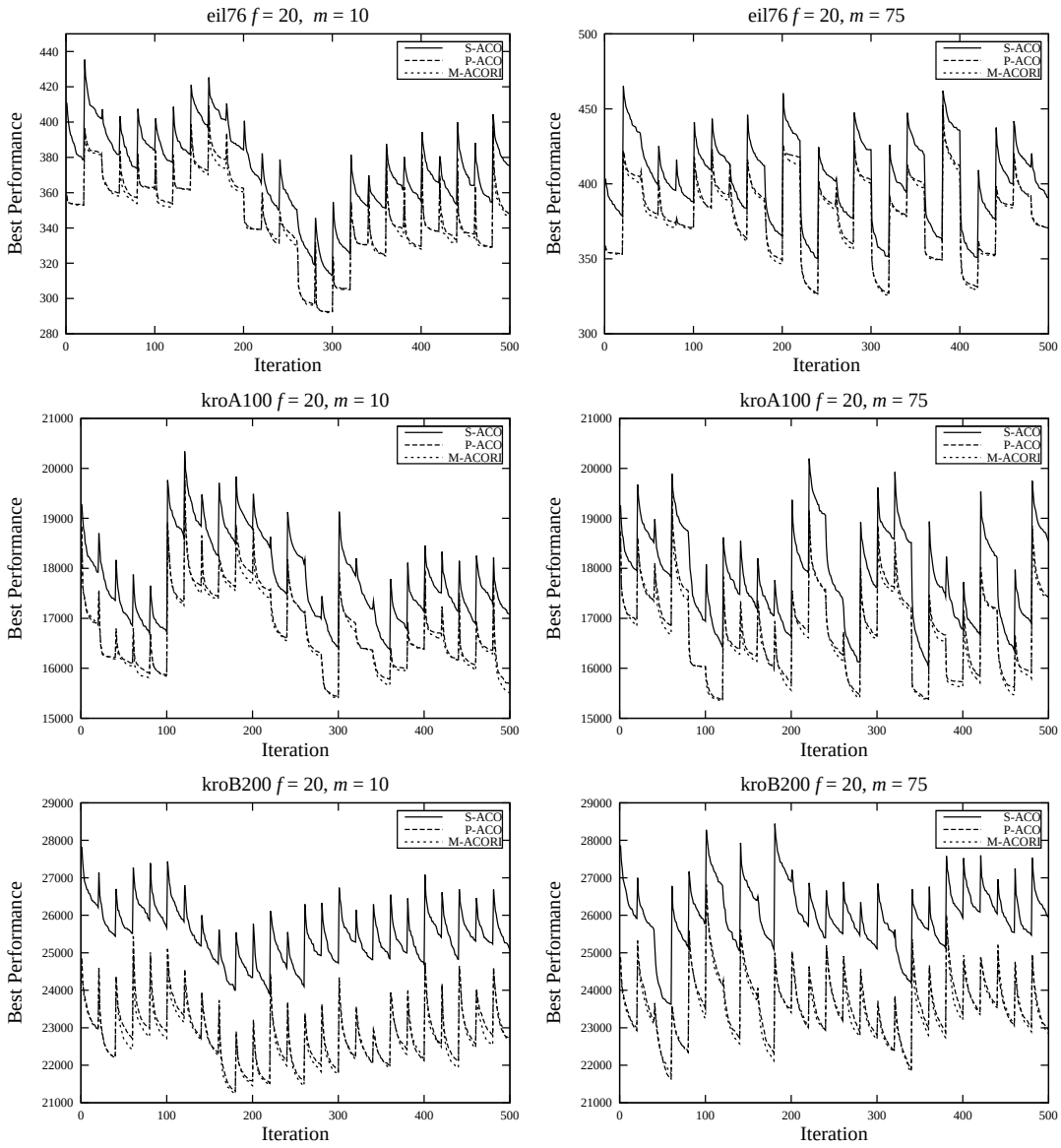

Fig. 5 Experiments regarding the overall *Best Performance* of M-ACORI with other ACO algorithms on dynamic environments.

ments are not similar. However, in our experiments, this is not the case since S-ACO is always significantly worse than the other algorithms. There are several reasons to explain this: (1) S-ACO pheromone trails are re-initialized with an equal amount of pheromone, without any guidance, (2) the ants stored in the population-list are repaired heuristically based on the new changing environment, using the $KeepElitist$ strategy [20], and the pheromone trails are re-initialized with guidance, (3) the initial experiments of comparing a traditional P-ACO and S-ACO with simple re-initialization strategy showed that they are competitive even when the changes are significant [14, pp. 264-265], and (4) apart from the guidance of the pheromone trails and the heuristic information, the AIO operator used in M-ACORI provides even more guidance to the population.

Third, regarding the overall *Average Performance* results, M-ACORI performs significantly worse than P-ACO. This is due to the fact that the LS operator in M-ACORI is applied only to the best ant in every iteration and not to the whole population. Therefore, only one ant improves its solution quality. In addition, the diversity scheme based on random immigrants avoids the population to converge into a single optimum. It is natural that the population may explore areas in the search space that are not advantageous for the current environment.

Fourth, as it was expected the S-ACO outperforms both P-ACO and M-ACORI on the *Average Performance* in slowly changing environments with significant changes because the ants stored in the population-list of P-ACO and M-ACORI do not seem to make much sense when the differ-

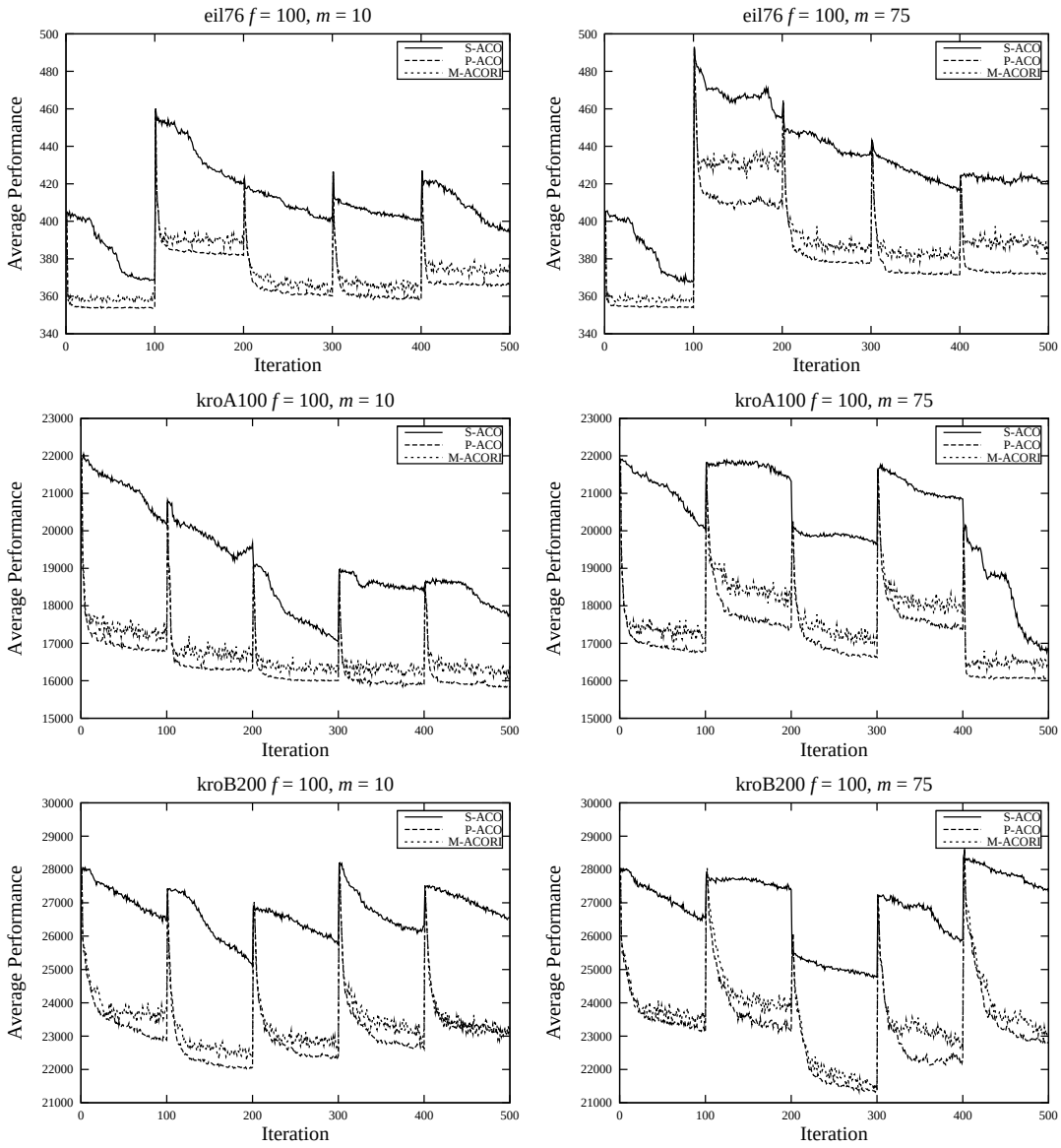

Fig. 6 Experiments regarding the overall *Average Performance* of M-ACORI with other ACO algorithms on dynamic environments.

ent environments are not similar. Therefore, using the restart strategy of S-ACO seems making more sense to start from scratch. Similarly, with the *Best Performance*, this is not the case in our experiments due to one of the possible reasons discussed above.

6.6 Experimental study regarding the overall robustness

In the fourth set of experiments, we study the overall robustness (both *Best Robustness* and *Average Robustness*) of M-ACORI in comparison with other ACO algorithms (S-ACO and P-ACO). The comparison results are given in Table 4, by the two-tailed t-test with 58 degree of freedom at a 0.05 level of significance, where "$s+$", "$s-$", and "$\sim$" have the same meaning as explained before. In addition, to visually observe the dynamic performance of the algorithms, the results regarding the *Best Robustness* and the *Average Robustness* are plotted in Figs. 7 and 8, respectively. For both figures, the two values of m, i.e., $m = 10\%$ and $m = 75\%$, are plotted, which represent slight and significant changes, respectively. The f value with a fast frequency of change, i.e., $f = 20$, is used in Fig. 7 for the *Best Robustness*, and with a slow frequency of change, i.e., $f = 100$, is used in Fig. 8 for the *Average Robustness*. From Table 4 and Figs. 7 and 8, several results can be observed.

First, in general, all algorithms have a high robustness level due to the use of the heuristic information, i.e., η, which is used in the probabilistic rule of ants in Eq. 1. In [63], it was discussed that performance and robustness can-

Table 4 Statistical test results regarding the overall robustness of comparing M-ACORI with other ACO algorithms.

Algorithms	eil76				kroA100				kroB200			
Best Robustness												
$f = 20, m \Rightarrow$	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
S-ACO⇔P-ACO	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
S-ACO⇔M-ACORI	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
P-ACO⇔M-ACORI	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
$f = 100, m \Rightarrow$	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
S-ACO⇔P-ACO	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
S-ACO⇔M-ACORI	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
P-ACO⇔M-ACORI	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
Average Robustness												
$f = 20, m \Rightarrow$	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
S-ACO⇔P-ACO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$\sim$	$s+$	$s+$	$s+$	$s+$	$s+$
S-ACO⇔M-ACORI	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
P-ACO⇔M-ACORI	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$s+$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
$f = 100, m \Rightarrow$	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
S-ACO⇔P-ACO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
S-ACO⇔M-ACORI	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
P-ACO⇔M-ACORI	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$

not be optimized simultaneously. This may be true for EAs, but not always true for ACO algorithms, especially ACO-based MAs, because of the prior knowledge they have beforehand and the extra knowledge from the LS operation. As a result, the population of ants cannot reach an extremely high diversity level, even with a global re-initialization of pheromone trails, to force the robustness to fall. This can be observed from the results where M-ACORI has a better performance than P-ACO while their robustness is not significantly different. For example, P-ACO and M-ACORI slightly sacrifice their *Best Robustness* for the sake of *Best Performance* whereas they significantly sacrifice their *Average Robustness* for the sake of *Average Performance*. This can be observed from the statistical test results in Tables 3 and 4.

Second, regarding the overall *Best Robustness* results, the proposed M-ACORI algorithm is slightly beaten by the other two algorithms on almost all problem instances, whereas S-ACO slightly beats P-ACO. Although M-ACORI improves the *Best Performance* in comparison with both P-ACO and S-ACO, they are not significantly different regarding the *Best Robustness*. Similarly, as it was expected S-ACO has better robustness than P-ACO and M-ACORI, since usually methods with worse performance have better robustness. This shows that S-ACO with simple re-initialization of pheromones may not be an effective method for DTSPs.

Third, regarding the overall *Average Robustness* results, S-ACO significantly beats the other two algorithms on all problem instances, either with slow or fast frequencies of change. Although LS in M-ACORI improves the *Average Performance* in comparison with S-ACO, the *Average Robustness* is degraded. This is expected because, as discussed previously, LS does not affect the whole population but only one ant. On the other hand, P-ACO significantly beats M-ACORI regarding *Average Performance*, but they are not significantly different regarding *Average Robustness*.

6.7 Experimental study under dynamic environments of varying change frequency and magnitude

We finally study the overall performance (both *Best Performance* and *Average Performance*) of M-ACORI in comparison with other ACO algorithms under dynamic environments with varying frequencies and magnitudes. In the basic experiments in Sect. 6.5, we have investigated the overall performance, both *Best Performance* and *Average Performance*, of compared algorithms under dynamic environments with different fixed values of f and m, respectively. That is, the values of f and m remain fixed throughout the execution of an algorithm. However, for real-world problems, both values of f and m may vary during the execution of an algorithm. In order to investigate the performance of our proposed M-ACORI algorithm in comparison with the existing S-ACO and P-ACO algorithms under more realistic environments, a larger benchmark TSP problem, i.e., `att532`, was used to construct the DTSP, and a spare pool of cities was generated as in the basic experiments. The values of f and m were both generated randomly within the interval of $[1, 100]$ and $[0, 100]$, respectively, as presented in Fig. 9, for 1,000 iterations over 30 runs, using the same parameters and performance measurements as in the basic experiments.

The experimental results with the corresponding statistical test results are presented in Table 5 and Fig. 10. In Table 5, "$s+$" indicates that the algorithm in the row is significantly better than the one in the column, "$s-$" indicates that

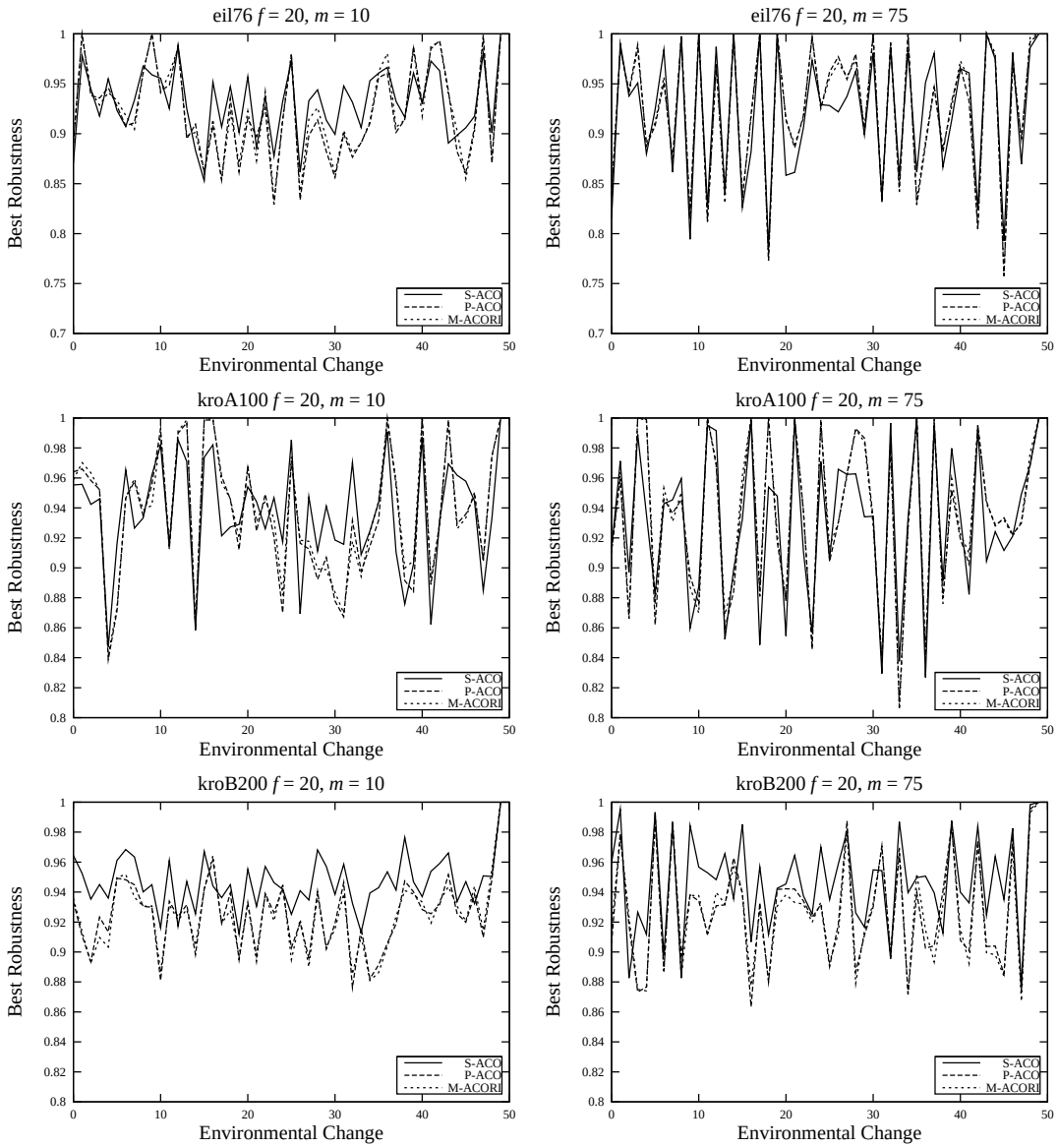

Fig. 7 Experiments regarding the overall *Best Robustness* of M-ACORI with other ACO algorithms on dynamic environments.

the algorithm in the column is significantly better than the one in the row, and "$\sim$" indicates statistically equivalent.

The results match our previous analysis regarding *Best Performance*, since M-ACORI continues to perform significantly better than its competitors. On the other hand, the results of the *Average Performance* do not match the results in our basic experiments. The difference is that the *Average Performance* of P-ACO is slightly better than M-ACORI, and not significantly better as analysed in our basic experiments previously. Even if their results are not significantly different, LS still does not affect the population as a whole, but only the best ant as discussed in the basic experiments.[4]

[4] The corresponding results of *Best Robustness* and *Average Robustness* for the environment with varying magnitudes and frequencies are similar with our basic experimental results.

The reason may be due to the distribution of cities in the specific TSP instance. If we take a closer look in Fig. 2, most of the cities in `att532` almost overlap, in comparison with the other instances used in the basic experiments. As a result, the distances between a city and its neighbourhood have more chances to be similar even after a dynamic change. Due to the heuristic information used from ACO algorithms when they construct their solutions, they have less chances to explore areas that may cause an increase to the *Average Performance*, as it is caused in the basic experiments with the other problem instances.

The S-ACO algorithm continues to perform significantly worse than both P-ACO and M-ACORI algorithms regarding both the *Best Performance* and *Average Performance*. Its small standard deviation, especially on the *Average Perfor-*

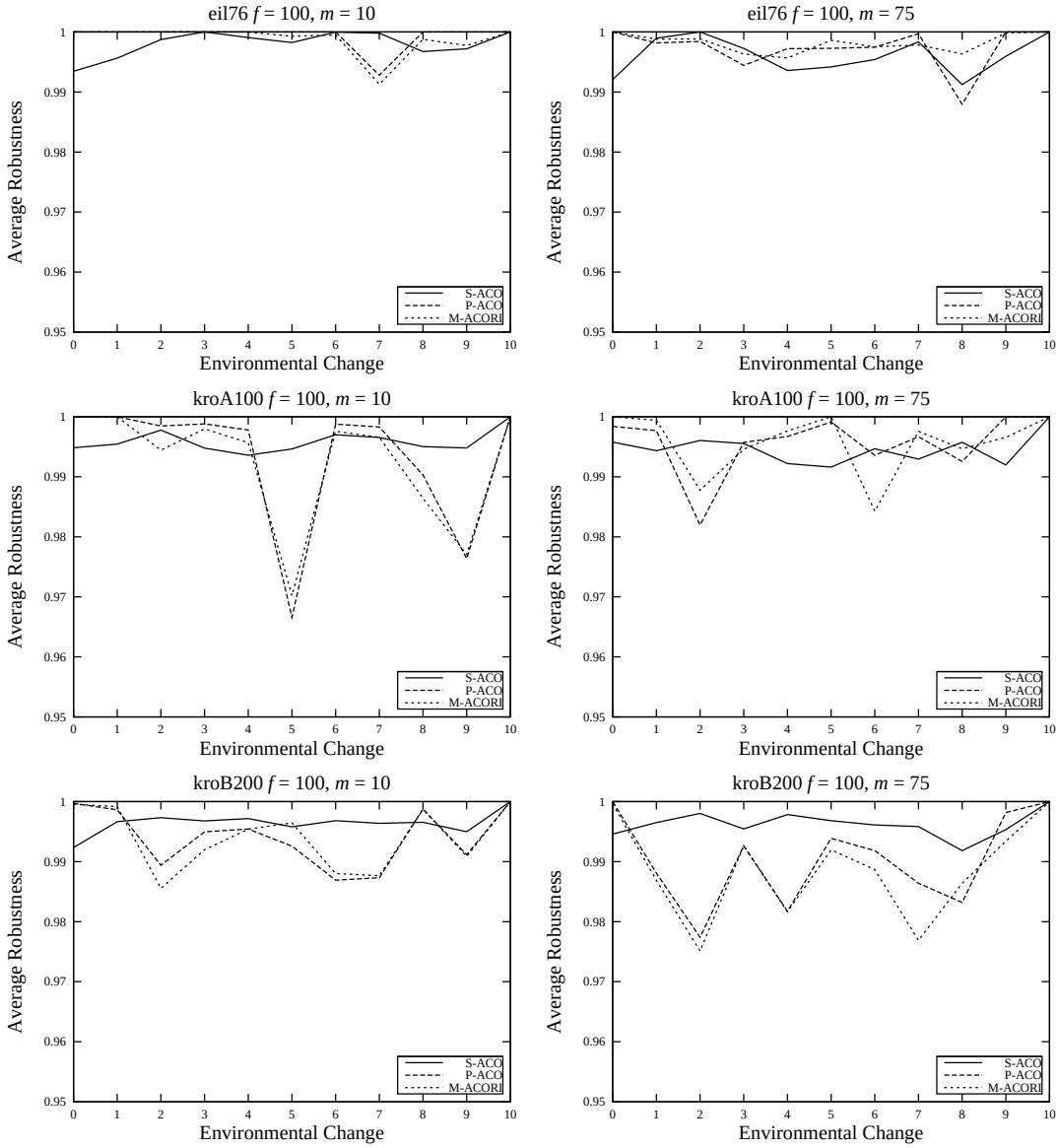

Fig. 8 Experiments regarding the overall *Average Robustness* of M-ACORI with other ACO algorithms on dynamic environments.

mance, indicates that S-ACO explores only a small area in the search space; see Table 5.

7 Conclusions and future work

In this paper, an ACO-based MA is developed, which uses an adaptive LS operator based on the IO operator inside an ACO algorithm with an enhanced population-list, to solve the DTSP. The component operators within the adaptive LS operator are activated adaptively in different periods of population evolution. The adaptive LS operator is used to further improve the quality of the solution obtained by the P-ACO algorithm with its strong exploitation. In addition, to maintain a sufficient diversity level within the population in order to adapt well to dynamic changes, random immigrants are generated as soon as the algorithm reaches stagnation behaviour. Several sets of experiments were carried out to compare the performance and robustness of M-ACORI with a number of existing ACO algorithms.

From the experimental results and analysis of the proposed M-ACORI algorithm in comparison with S-ACO and P-ACO on the dynamic test problems, several concluding remarks can be drawn.

First, for ACO algorithms on DTSPs, especially ACO-based MAs, it is not always the case that performance and robustness cannot be optimized simultaneously as with EAs [63]. This is due to the prior knowledge ACO algorithms gain with the use of the heuristic information and the extra knowledge from the LS operation.

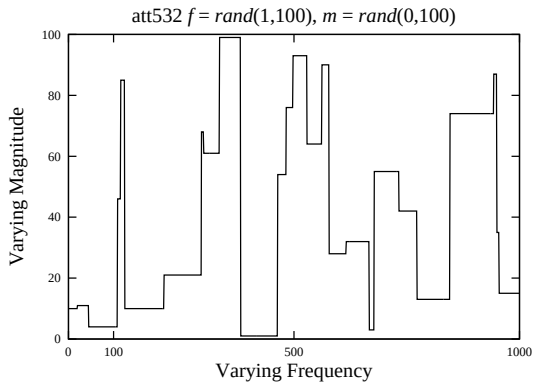

Fig. 9 The varying values of f and m generated for the dynamic environment with varying dynamics, which are used in Fig. 10. Note that the value of f is calculated with the iterations.

Table 5 Experimental results of comparing the overall performance of M-ACORI with other ACO algorithms on the dynamic `att532` problem with varying change frequency and magnitude

Algorithms	S-ACO	P-ACO	M-ACORI
Best Performance (Standard Deviation)			
	75276.2(156.7)	68038.4(221.4)	67839.2(307.7)
t-Test Result			
S-ACO		$s-$	$s-$
P-ACO	$s+$		$s-$
M-ACORI	$s+$	$s+$	
Average Performance (Standard Deviation)			
	85152.6(19.7)	72622.9(351.5)	72662(416.9)
t-Test Result			
S-ACO		$s-$	$s-$
P-ACO	$s+$		$\sim$
M-ACORI	$s+$	$\sim$	

Second, the proposed M-ACORI algorithm with the use of the diversity scheme improves the solution quality of the M-ACO algorithm without the diversity scheme. The combination of ACO with an LS operator provides strong exploitation which may not be advantageous when addressing DOPs. Therefore, ACO-based MAs enhanced with the diversity scheme appropriately, i.e., introducing random immigrants once the population has converged, leads the algorithm to achieving better *Best Performance*; otherwise, a high level of diversity may lead to randomization, and not always to good results.

Third, the proposed AIO operator promotes cooperation of different operators that can be advantageous on different test problems or during different periods of population evolution. It is known that different operators are problem dependent. However, the AIO operator helps M-ACORI enhance its convergence speed and solution quality simultaneously.

Fourth, although M-ACORI improves the *Best Performance*, it maintains its robustness, since the *Best Robustness* from the experimental results shows no significant difference in comparison with the other ACO algorithms.

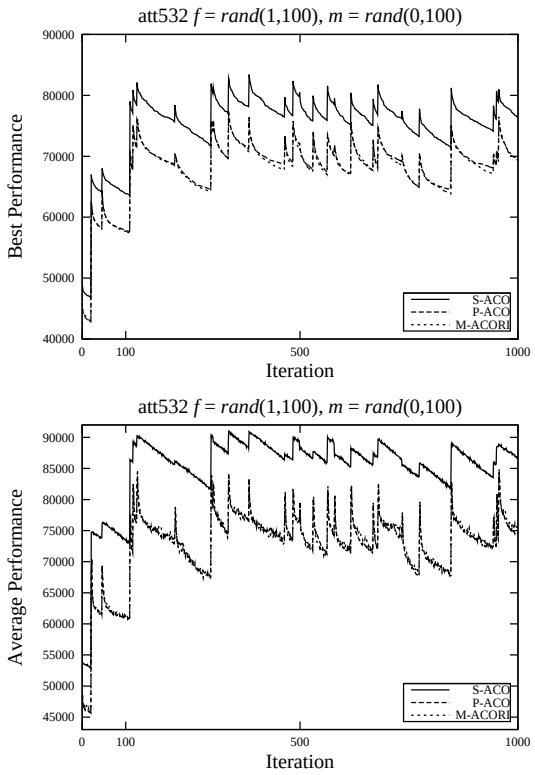

Fig. 10 Experimental results regarding the *Best Performance* and *Average Performance* of M-ACORI in comparison with other ACO algorithms on the dynamic `att532` problem with varying change frequency and magnitude.

Fifth, the *Average Performance* depends on the distribution of cities of the stationary TSP instance used to generate DTSPs. It is natural to have an increase in the *Average Performance* when a certain level of diversity is always maintained in the population because some ants within the population have more chances to locate areas that may not be advantageous for the distribution of the population as a whole, but it may be advantageous for the best ant of the new environment. However, when too many cities overlap on their distribution, the population has less chances to explore new areas since the new environment generated has more chances to be similar with the previous one.

Finally, the S-ACO algorithm with the restart strategy is significantly worse on all dynamic test cases, regarding both *Best Performance* and *Average Performance*. However, S-ACO is significantly better on all dynamic test cases, regarding *Average Robustness*. S-ACO explores only a small area in the search space which makes the population more stable as a whole, but not so consistent with its best output.

For future work, the proposed M-ACORI framework is a good start point for developing other ACO-based MAs for DTSPs. Since multiple LS operators are effective with a diversity scheme, it would be interesting to apply other more sophisticated methods [32] as LS operators, develop other

diversity schemes, and hybridize them. Also, it would be interesting to apply LS to the whole population-list of an ACO-based MA. This may also affect the population as a whole and improve the *Average Performance* and *Average Robustness*.

Acknowledgements The authors would like to thank the guest editors and anonymous reviewers for their thoughtful suggestions and constructive comments. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E060722/01 and Grant EP/E060722/02.

References

1. Acampora G, Loia V, Gaeta M (2010) Exploring e-learning knowledge through othological memetic agents. IEEE Comput Intell Mag 5(2):66-77
2. Angus A (2006) Niching for population-based ant colony optimization. In: Proceedings of the 2nd IEEE international conference on e-science and grid computing, pp 15–22
3. Bi X, Luo G (2007) The improvement of ant colony algorithm based on the inver-over operator. In: Proceedings of the International Conference on Mech and Autom, pp 2383–2387
4. Bullnheimer B, Hartl RF, Strauss C (1999) A new rank based version of the Ant System - A computational study Tech Rep, Institute of Management Science, University of Vienna, Austria
5. Branke J (1999) Memory enhanced evolutionary algorithms for changing optimization problems. In: Proceedings of the 1999 IEEE congress on evolutionary computation, pp 1875–1882
6. Branke J, Kauβler T, Schmidth C, Schmeck H (2000) A multi-population approach to dynamic optimization problems. In: Proceedings of the 4th international conference on adaptive computing in design and manufacturing, pp 299-308
7. Branke J, Salihoğlu E, Uyar Ş (2005) Towards an analysis of dynamic environments. In: Proceedings of the 2005 Gen and Evol Comput Conf, pp 1433–1400
8. Bonabeau E, Dorigo M, Theraulaz G (1997) Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, New York
9. Caponio A, Cascella GL, Neri F, Salvatore N, Summerr M (2007) A fast adaptive memetic algorithm for online and offline control design of PMSM drives. IEEE Trans on Syst Man and Cyber B Cyber 37:28-41
10. Colorni A, Dorigo M, Maniezzo V (1992) Distributed optimization by ant colonies. In: Proceedings of the 1st European Conf on Artificial Life, pp 134–142
11. Di Caro G, Dorigo M (1998) Ant Net: Distributed Stigmergetic Control for Communications Networks. J of Artif Intell Res 9: 317–365
12. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans on Evol Comput 1(1):53–66
13. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans on Systems Man and Cybern B Cybern 26(1):29–41
14. Dorigo M, Stützle T (2004) Ant colony optimization. The MIT Press, London, England
15. Eyckelhof CJ, Snoek M (2002) Ant systems for a dynamic TSP. In: Proceedings of the 3rd Int Workshop on Ant Algorithms, pp 88–99
16. Eriksson R, Olsson B (2002) On the behaviour of evolutionary global-local hybrids with dynamic fitness functions. In: Proceedings of the 7th international conference on parallel problem solving from nature, pp 13–12
17. Eriksson R, Olsson B (2004) On the performance of evolutionary algorithms with life-time adaptation in dynamic fitness landscapes. In: Proceedings of the 2004 IEEE congress on evolutionary computation, pp 1293–1300
18. Isaacs A, Puttige V, Ray T, Smith W, Anavatti S (2008) Development of a memetic algorithm for dynamic multi-objective optimization and its application for online neural network modeling of UAVs. IEEE international joint conference on neural networks (IJCNN 2008), pp 548–554
19. Ishibuchi H, Yoshida T, Murata T (2003) Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE Trans Evol Comput 7(2):204–233
20. Guntsch M, Middendorf M (2002) Applying population based ACO to dynamic optimization problems. In: Proceedings of the 3rd Int Workshop on Ant Algorithms, LNCS 2463, pp 111–122
21. Guntsch M, Middendorf M (2002) A population based approach for ACO. In: EvoWorkshops 2002: Appl of Evol Comput, pp 72–81
22. Guntsch M, Middendorf M (2001) Pheromone modification strategies for ant algorithms applied to dynamic TSP. In: EvoWorkshops 2001: Appl of Evol Comput, pp 213–222
23. Guntsch M, Middendorf M, Schmeck H (2001) An ant colony optimization approach to dynamic TSP. In: Proceedings of the Gen and Evol Comput Conf, pp 860–867
24. Guo T, Michalewicz Z (1998) Inver-over operator for the TSP. In: Proc of the 5th International Conference on Parallel Problem Solving from Nature, pp 803–812
25. Grefenestette JJ (1992) Genetic algorithms for changing environments. In: Proceedings of the 2nd international conference on Parallel Problem Solving from Nature, pp 137–144
26. Jen E (2003) Stable or robust? What's the difference? Complexity 8(3):12–18
27. Jin Y, Branke J (2005) Evolutionary optimization in uncertain environments - a survey. IEEE Trans Evol Comput 9(3):303–317
28. Krasnogor N, Smith J (2005) A tutorial for competent memetic algorithms: model, taxonomy, and design issues. IEEE Trans Evol Comput 9(5):474–487
29. He J, Yao X (2002) From an Individual to a population: an analysis of the first hitting time of population-bases evolutionary algorithms. IEEE Trans Evol Comput 6(5):495–511
30. Holland J (1975) Adaption in Natural and Artificial Systems, University of Michigan Press
31. Lee Z-J, Su S-F, Chuang C-C, Liu K-H (2006) Genetic algorithm with ant colony optimization for multiple sequence alignment. Appl Soft Comput 8(1):55-78
32. Lin S, Kerninghan BW (1972) An effective heuristic algorithm for the traveling salesman problem. Oper Res 21:498–516
33. Lim D, Ong Y-S, Jin Y, Sendhoff B, Lee BS (2007) Efficient hierarchical parallel genetic algorithms using grid computing. Future Gener Comput Syst 23(4):658–670
34. Lim D, Ong Y-S, Lim MH, Jin Y (2007) Single/Multi-objective inverse robust evolutionary design methodology in the presence of uncertainty. Evolutionary computation in dynamic and uncertain environments, Series on studies in computational intelligence 51: 437–456
35. Lim KK, Ong Y-S, Lim MH, Chen X, Agarwal A (2008) Hybrid ant colony algorithms for path planning in sparse graphs Soft Comput 12(10):981–994
36. Liu B, Wang L, Jun YH (2007) An effective PSO-based memetic algorithm for flow shop scheduling. IEEE Trans Syst Man Cybern B Cybern 37(1):18–27
37. Mavrovouniotis M and Yang S (2010) Ant colony optimization with immigrants schemes for dynamic environments. In: Proceedings of Parallel Problem Solving from Nature PPSN XI, pp 371–380

38. Mei Y, Tang K, Yao X (2009) Improved memetic algorithm for capacitated arc routing problem. In: IEEE congress on evol comput, pp 1699–1706
39. Michalewicz Z (1999) Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, Berlin, third edition
40. Morrison RW (2003) Performance measurement in dynamic environments. GECCO workshop on evolutionary algorithms for dynamic optimisation problems, pp 5–8
41. Neumann F, Witt C (2009) Runtime analysis of a simple ant colony optimization algorithm. Algorithmica 54(2):243–255
42. Neri F, Toivanen J, Cascella GL, Ong Y-S (2007) An adaptive multimeme algorithm for designing HIV multidrug therapies. IEEE/ACM Trans Comput Biol Bioinform 4(2):264–278
43. Neri F, Mininno E (2010) Memetic compact differential evolution for cartesian robot control. IEEE Comput Intell Magaz 5(2):54–65
44. Ong Y-S, Keane AJ (2004) Meta-lamarckian learning in memetic algorithms, IEEE Trans Evol Comput 8(2):99–110
45. Ong Y-S, Lim MH, Chen X (2010) Research frontier: memetic computing - past, present and future. IEEE Comput Intell Magaz 5(2):24–36
46. Ong Y-S, Lim MH, Zhu N, Wong KW (2006) Classification of adaptive memetic algorithms: a comparative study. IEEE Trans Syst Man Cybern B Cybern 36(1):141–152
47. Raman N, Talbot FB (1993) The job shop tardiness problem: a decomposition approach. Eur J Oper Res 69(2):187–199
48. Rand W, Riolo R (2005) Measurements for understand the behaviour of the genetic algorithm in dynamic environments: a case study using the shaky ladder hyperplane-defined functions. In: Proceedings of the 2005 genetic and evolutionary computation conference, pp 32–38
49. Smith JE (2007) Coevolving memetic algorithms: a review and progress report. IEEE Trans Syst Man Cybern B Cybern 37(1):6–17
50. Stützle T, Hoos H (1997) The MAX-MIN ant system and local search for the traveling salesman problem. In: Proceedings of the 1997 IEEE international conference on Evol Comput, pp 309–314
51. Stützle T, Hoos H (2000) MAX-MIN ant system. Fut Gen Comput Syst 8(16):889–914
52. Talbi EG, Bachelet V (2006) Cosearch: a parallel cooperative metaheuristic. J Math Model Algorithms 5(1):5–22
53. Tang K, Yao X (2007) Memetic algorithm for VLSI floorplanning, IEEE Trans Syst Man Cybern B Cybern 37(1):62–69
54. Tang K, Mei Y, Yao X (2009) Memetic algorithm with extended neighborhood search for capacitated arc routing problems. IEEE Trans Evol Comput 13(5):1151–1166
55. Wang H, Wang D, Yang S (2009) A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. Soft Comput 13(8-9):763–780
56. Wang J, Osigie E, Thulasiraman P, Thulasiram RK (2009) HOPNET: A hybrid ant colony optimization routing algorithm for mobile ad hoc network. Ad Hoc Networks 7(4):690–705
57. Weicker K (2002) Performance measures for dynamic environments. In: Proceedings of Parallel Problem Solving from Nature PPSN VII, pp 64–73
58. William EH, Krasnogor N, Smith JE (eds) (2005) Recent advances in memetic algorithms. Springer, Berlin
59. Xing L-N, Chen Y-W, Yang K-W (2008) A hybrid approach combining an improved genetic algorithm and optimization strategies for the asymetric traveling salesman problem. Engineering Appl of Artif Intell 21(8):1370-1370
60. Xing L-N, Chen Y-W, Yang K-W (2008) Double layer ant colony optimization for multi-objective flexible job shop scheduling problems. New Gener Comput 26(4):313–327
61. Xing L-N, Chen Y-W, Wang P, Zhao Q, Xiong J (2010) A knowledge-based ant colony optimization for flexible job shop scheduling problems. Appl Soft Comput 10(3):888–896
62. Yang S (2008) Genetic algorithms with memory and elitism based immigrants in dynamic environments. Evol Comput 16(3), pp 385–416
63. Yu X, Tang K, Chen T, Yao X (2009) Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization. Memetic Computing 1(1):3–24
64. Zhang X, Tang L (2008) A new hybrid ant colony optimization algorithm for the vehicle routing problem. Pattern Recognition Letters 30(9):848–855
65. Zhou Z, Ong Y-S, Lim MH (2007) Memetic algorithm using multi-surrogates for computationally expensive optimization problems. Soft Comput 11(9):873-888