Ant Colony Optimization With Local Search for Dynamic Traveling Salesman Problems

Michalis Mavrovouniotis, *Member, IEEE*, Felipe M. Müller, and Shengxiang Yang, *Senior Member, IEEE*

***Abstract*—For a dynamic traveling salesman problem (DTSP), the weights (or traveling times) between two cities (or nodes) may be subject to changes. Ant colony optimization (ACO) algorithms have proved to be powerful methods to tackle such problems due to their adaptation capabilities. It has been shown that the integration of local search operators can significantly improve the performance of ACO. In this paper, a memetic ACO algorithm, where a local search operator (called unstring and string) is integrated into ACO, is proposed to address DTSPs. The best solution from ACO is passed to the local search operator, which removes and inserts cities in such a way that improves the solution quality. The proposed memetic ACO algorithm is designed to address both symmetric and asymmetric DTSPs. The experimental results show the efficiency of the proposed memetic algorithm for addressing DTSPs in comparison with other state-of-the-art algorithms.**

***Index Terms*—Ant colony optimization (ACO), dynamic traveling salesman problem (DTSP), local search, memetic algorithm.**

I. Introduction

THE TRAVELING salesman problem (TSP) is one of the most fundamental $\mathcal{NP}$-complete combinatorial optimization problems [20]. The classic TSP can be described as follows: given a collection of cities, the objective is to find the shortest Hamiltonian cycle that starts from one city and visits each of the other cities once before returning to the starting city. Over the years, exact methods [8], heuristics [35], [55] and metaheuristics [13], [57] have been proposed to solve the static TSP. Due to the strong increase in the computation time when the problem size increases, heuristics and metaheuristics are more preferable than exact methods since they trade optimality for efficiency.

Manuscript received November 8, 2015; revised February 16, 2016, April 11, 2016, and April 15, 2016; accepted April 16, 2016. Date of publication June 13, 2016; date of current version June 14, 2017. This work was supported in part by the Engineering and Physical Sciences Research Council of U.K. under Grant EP/K001310/1, and in part by the Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior, Brazil, under Grant BEX 2380/14-5. This paper was recommended by Associate Editor M. Dorigo. *(Corresponding author: Shengxiang Yang.)*

M. Mavrovouniotis and S. Yang are with the Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University, Leicester LE1 9BH, U.K. (e-mail: mmavrovouniotis@dmu.ac.uk; syang@dmu.ac.uk).

F. M. Müller is with the Technological Center, Department of Applied Computing, Federal University of Santa Maria, Santa Maria 97105-900, Brazil (e-mail: felipe@inf.ufsm.br).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org provided by the authors. This includes the details of the parameter tuning for the evaporation rate used within the proposed algorithm. This material is 39.8 KB in size.

Digital Object Identifier 10.1109/TCYB.2016.2556742

In the last decade, there is an increasing interest to address dynamic versions of the TSP, such as the dynamic TSP (DTSP) where the topology of cities changes [1], [23], [24], [31], [32] or the weights between the cities change [16], [41], [44], [51]. These kind of problems have more practical value [26], [45]. For instance, a vehicle of a small company needs to distribute items sold to different customers starting from and returning to his base again after all the customers are satisfied. The task is to optimize the time and plan a route as efficiently as possible. Therefore, by considering the travel time between customers it can generate the route and start. However, the travel time between two customers may change during its route due to traffic jams or any other factors. Hence, the route needs to be reoptimized considering the new factors.

Basically, a DTSP can be viewed as a sequence of different static TSP instances that change over time. If the time interval between the changes is long, then a straightforward way is to apply exact methods, e.g., Concorde [8], or effective heuristics, e.g., Lin–Kernighan [35], 2-Opt [9], 3-Opt [36], to reoptimize whenever a dynamic change occurs, assuming that the changes are detectable. Such methods are capable of finding the global optimum (or close to the global optimum) solution for symmetric TSP cases in seconds. In contrast, the field of evolutionary dynamic optimization offers several metaheuristics that tackle DTSPs [10], [59] by using knowledge transferred from previously optimized instances. Such methods are suitable to speed up the reoptimization process when the changes are small to medium [6], [27]. Furthermore, they are more appropriate in cases where the time interval between the changes is relatively short, which makes exact or any other computationally expensive methods inappropriate [56].

In fact, the change interval in many real-world applications of DTSPs, e.g., in logistics [50], [52], robotics [22], and telecommunications [2], is rather short. Another important characteristic of such applications is that the dynamic changes are usually asymmetric. Most exact methods, e.g., Concorde, and heuristic methods, e.g., Lin–Kernighan, 2-Opt or 3-Opt, are not (directly) applicable for asymmetric TSP cases since they are based on the triangle inequality of Euclidean distance associated with symmetric TSPs. Methods for asymmetric TSPs are less studied with only a few exceptions [30], [53]. Of course, symmetric methods can be applied with some modifications to the problem or the method itself. But, this may significantly increase the computation time or degrade their effectiveness [29], [52].

Among the metaheuristics developed in the evolutionary dynamic optimization domain, ant colony optimization (ACO) [12]–[15] algorithms are extensively used to address (symmetric) DTSPs due to their adaptation capabilities. More precisely, the pheromone trails generated before a dynamic change can be used after a change to speed up reoptimization. Different strategies were integrated into ACO to maintain a high quality of output efficiently [16], [23], [24], [41], [44]. Recently, a local search operator was integrated into an ACO-based memetic algorithm [38]. Local search algorithms can better explore locally a neighborhood in the search space, and together with the adaptation capabilities of ACO, the resulting ACO-based memetic algorithm serves as a powerful algorithm for the DTSP. The pheromone update policy of the algorithm is based on one of the best performing ACO algorithms, i.e., $\mathcal{MAX}$–$\mathcal{MIN}$ ant system (AS) ($\mathcal{MM}$AS) [53]. The local search operator used is the unstringing and stringing (US) operator [21]. $\mathcal{MM}$AS provides its best solution to the US operator for local search improvements before the pheromone update procedure.

The memetic algorithm, denoted as $\mathcal{MM}\text{AS}_{\text{US}}$, performs two node insertion and two node removal moves and assumes that the DTSP is symmetric [38]. However, it was not designed with the triangle inequality assumption as with other local search operators (e.g., 2-Opt and 3-Opt). In this paper, the $\mathcal{MM}\text{AS}_{\text{US}}$ algorithm [38] is extended to also cope with asymmetric DTSPs. The major contributions of this paper are summarized below.

1) A memetic framework based on ACO is proposed that triggers local search optimization whenever a new best solution is discovered.
2) Two additional node insertion moves and two additional node removal moves are proposed for the DTSP that improve the performance in the asymmetric cases without affecting the performance in the symmetric cases.
3) A DTSP benchmark generator that encompasses some real-world applications is used to systematically investigate the performance of $\mathcal{MM}\text{AS}_{\text{US}}$.
4) The DTSP benchmark generator with weight changes in [41] is extended to also generate asymmetric dynamic changes. Although an asymmetric dynamic change is a real-world characteristic, it has received less attention from researchers. In this paper, asymmetric dynamic changes are extensively studied.

The rest of this paper is organized as follows. Section II introduces the DTSP problem definition, different variations of DTSPs, and describes how the DTSP cases are generated. Section III describes the methods from the literature that have been applied to different variations of DTSPs. Section IV gives details of the proposed $\mathcal{MM}\text{AS}_{\text{US}}$ algorithm. Section V presents the experimental studies that include comparisons with other popular heuristics and state-of-the-art algorithms. Finally, concluding remarks and future work directions are presented in Section VI.

II. Dynamic Traveling Salesman Problems

A. Problem Formulation

Typically, a TSP instance is modeled by a fully connected weighted graph $G = (N, A)$, where $N = \{v_1, \ldots, v_n\}$ is a set of n nodes and $A = \{(v_i, v_j) \mid v_i, v_j \in N, i \neq j\}$ is a set of arcs. For the classic TSP, nodes and arcs represent the cities and the links between them. Each arc $(v_i, v_j) \in A$ is associated with a non-negative value $d_{ij} \in \mathbb{R}^+$, which for the classic TSP represents the distance or travel time between cities v_i and v_j. A TSP instance is considered to be symmetric if $d_{ij}(t) = d_{ji}(t), \forall (v_i, v_j) \in A$, or asymmetric if $d_{ij}(t) \neq d_{ji}(t)$ for at least one $(v_i, v_j) \in A$. Often, an undirected or directed graph G is used for a symmetric or asymmetric TSP instance, respectively. The asymmetric TSP appears to be more challenging to solve than the symmetric TSP because an algorithm needs to consider the direction of the arcs as well [7], [28]. The most studied and well known special case of symmetric TSP is the version in which the distances among cities always satisfy the triangle inequality (e.g., $d_{ij} + d_{jk} \geq d_{ik}$), known as Euclidean TSP. There are many existing methods, both exact and heuristic, that are based on the triangle inequality [8], [35]. However, the corresponding asymmetric special case may not always satisfy the triangle inequality.

Considering that all undirected graphs can be viewed as directed graphs (e.g., by duplicating the arcs in both directions), then a symmetric TSP can be considered as a special case of the asymmetric TSP. In contrast, it is also possible to transform an asymmetric TSP instance to a symmetric one by doubling the number of nodes [29]. The symmetric version of TSP is one of the most studied versions of TSP in the literature, whereas the asymmetric version has attracted less attention [7]. For instance, Gerhard Reinelt's TSP library (TSPLIB)[1] offers much fewer and smaller problem instances for the asymmetric TSP in comparison with the symmetric TSP.

The TSP, either symmetric or asymmetric, becomes more realistic and challenging if it is subject to a dynamic environment. Specifically, for the DTSP where the weight matrix is subject to changes, it can be defined as follows:

$$\mathbf{D}(t) = \left\{d_{ij}(t)\right\}_{n\times n} \tag{1}$$

where t is the period of a dynamic change. A particular solution $s = [s_1, \ldots, s_n]$ in the search space is specified by a permutation of the nodes, and for the DTSP, it is evaluated as follows:

$$f(s, t) = d_{s_n s_1}(t) + \sum_{i=1}^{n-1} d_{s_i s_{i+1}}(t). \tag{2}$$

B. Dynamic TSP Benchmark Generators

The concept of DTSPs was initially introduced by Psaraftis [47]. Since then, several variations of DTSPs were introduced, where the set of nodes N [1], [23], [24], [31], [32], [56] and/or the cost

[1] A library that consists of TSP problem instances with their optimal solutions, which is available at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

from the set of arcs A [16], [41], [44], [51], [56] cause the weight matrix $\mathbf{D}(t)$ to change during the optimization process. However, there is still no any unified benchmark problem for DTSPs, which makes the comparison with algorithms from the literature a very challenging task. One popular benchmark is the DTSP where cities are exchanged: half of the cities from the problem instance are removed to create a spare pool [23], [24], [39], and the cities from the spare pool are then used to replace cities from the problem instance. Another popular benchmark is the DTSP where the weights of arcs change probabilistically [41] (the complete benchmark generator description is given in Section II-C). In [16] and [51], only the weights of arcs that belong to the best tour increase or decrease accordingly.

Younes *et al.* [60] introduced a benchmark generator for the DTSP with different modes: 1) topology change as in [24]; 2) weights change as in [16]; and 3) swap cities. Based on the last mode (i.e., swap cities) of the aforementioned benchmark generator, a general dynamic benchmark generator for permutation-encoded problems (DBGP) was proposed that can generate test cases with known optima [42]. DBGP can convert any stationary TSP instance into a DTSP with specific properties (i.e., frequency and magnitude of changes). Although with DBGP one can observe how close to the optimum an algorithm converges, it sacrifices real-world models for the sake of benchmarking.

Since DBGP was used for the preliminary analysis of $\mathcal{MM}\text{AS}_{\text{US}}$ in [38] and showed that the algorithm recovers relatively close to the global optimum (for symmetric cases), in this paper we consider a benchmark generator that models real-world situations (i.e., the DTSP where weights change [41]) to further investigate the proposed $\mathcal{MM}\text{AS}_{\text{US}}$.

C. Generating Dynamic Test Cases

Considering the problem formulation above, a dynamic test case of a TSP can be generated by modifying the value of the arc between nodes v_i and v_j as follows:

$$d'_{ij} \leftarrow d_{ij} \times t_{ij} \tag{3}$$

where t_{ij} represents the change on the link between nodes v_i and v_j, which is generated as follows:

$$t_{ij} = \begin{cases} t_{ij} \leftarrow 1 + r \in [F_L, F_U], & \text{if } q \leq m \\ t_{ij} \leftarrow 1, & \text{otherwise} \end{cases} \tag{4}$$

where r is a random variable uniformly distributed in $[F_L, F_U]$, where F_L and F_U define the lower and upper bounds of the change, respectively, q is a random variable uniformly distributed in $[0, 1]$, and m defines the magnitude of change that satisfies $0 < m \leq 1$. For every arc, a different r value is generated to embed real-world characteristics in the constructed DTSP. In this way, links with higher changes are generated when r values are closer to F_U, or links with smaller changes are generated when r values are closer to F_L. For example, traffic jams are not the same in every street. Similarly, delays are not the same in all the links where packets are sent in communication networks. If t_{ij} is set to 1, it indicates that there is no change between nodes v_i and v_j. Although r may define the degree of a change in a single arc individually, the overall magnitude of change is expressed by the number of arcs that will change, i.e., m.

So far, only the magnitude of change was discussed. The frequency of change defines how quickly changes will occur. For this DTSP, the frequency f of change is synchronized with the algorithm: every f algorithmic iterations/evaluations, a change occurs as defined in Eq. (4).

Since many real-world problems can be formulated as DTSPs and methods for solving static TSPs can be applied to solve them [7], [22]; the dynamic changes generated in this paper can be generalized and may represent different factors depending on the application. For example, in logistics, they may represent traffic on the road system or in telecommunications they may represent delays on the network. Therefore, the factor t_{ij} in Eq. (3), i.e., traffic jam, is multiplied with the normal travel time or physical distance of the road, i.e., d_{ij}. In this way, the factor is normalized according to the scale of the d_{ij} values. In another DTSP benchmark generator [56], the dynamic changes are generated with the addition of factor t_{ij} generated from a normal distribution (also negative values are allowed) rather than a uniform distribution (only positive values are allowed) as in our case. Such dynamic changes make more sense for real-world problems where the original weights are also possible to be deducted.

Furthermore, the existing benchmark generator has only been used to generate symmetric cases of DTSPs [39]. For example, the dynamic changes generated between nodes v_i and v_j satisfy the following policy: $t_{ij} = t_{ji}$. It is straightforward that from the same problem instance a corresponding asymmetric case of the DTSP can be generated in which the dynamic change of one direction t_{ij} may be different from the opposite direction t_{ji}. In fact, such case is closer to a real-world scenario, e.g., the traffic jams on the road system are not necessarily the same in both directions.

III. Methodologies for Solving DTSPs

The challenges of algorithms in solving $\mathcal{NP}$-complete problems in static environments are well-known in terms of computational complexity [20]. But, addressing $\mathcal{NP}$-complete problems in dynamic environments is even more challenging for algorithms because the objective is not only to locate the global optimum efficiently, but also to track it over the environmental changes. This requires repeated optimization whenever a dynamic change occurs. In the following sections, we briefly review existing algorithms designed to address different variations of DTSPs.

A. ACO Algorithms

Angus and Hendtlass [1] applied a conventional ACO algorithm to solve a small DTSP instance (e.g., Burma14 with 14 cities), where a single city was removed during the execution. They observed that reoptimizing via adaptation in ACO is faster than a complete restart.

Eyckelhof and Snoek [16] considered a different DTSP where the weights between cities change with time, representing sudden changes in the traffic.

A "shaking" technique was proposed to the basic AS, where the pheromone trails are smoothed after a dynamic change.

Guntsch and Middendorf [23] proposed local restart strategies for the DTSP where the topology of cities changes. The idea is that, instead of reinitializing all the pheromone trails, the pheromone trails of the cities affected by the dynamic changes, i.e., inserted or removed cities, are reinitialized heuristically. Later on, they also introduced one of the most studied ACO for the same DTSP, known as the population-based ACO (P-ACO), where no pheromone evaporation is used [24]. The framework consists of a memory of limited size, where the best ant of every iteration is added. The memory uses a first-in-first-out policy. When the dynamic changes affect the ants stored in the memory, they are repaired heuristically using a keep-elitist strategy [25]. The pheromone trails are updated according to the solutions currently stored in the memory. An improvement of the P-ACO was proposed in [39], denoted as memetic P-ACO, where simple and adaptive inversions [55] were adaptively applied to the best ant before it is added to the memory.

Immigrants schemes were integrated with ACO to address the DTSP with weight changes [41]. The idea is to generate immigrant ants that will deposit pheromone, either randomly, e.g., in random immigrants ACO, or in a guided way, e.g., in elitism-based immigrants ACO (EIACO). Furthermore, multicolony approaches were also applied to the same DTSP. For example, the multicolony schemes proposed in [43] use a separate pheromone table for each colony whereas the multicolony schemes proposed in [44] consist of a single colony with several castes (i.e., a group of ants that have the same behavior but different from other groups of ants) that use the same pheromone table. The 3-Opt local search is applied to improve solution quality in all castes. The colonies/castes use different exchange policies to communicate.

B. Evolutionary Algorithms

Zhou *et al.* [61] integrated three operators, i.e., insert, delete and change, to the inver-over algorithm [55] to cope with the insertion, deletion, and modification, respectively, of cities in TSPs. Li *et al.* [32] further improved the inver-over algorithm using a pool that maintains a set of the most promising gene segments by applying several heuristic rules. The algorithm was applied to the DTSP that is based on the CHN146+3 benchmark [31] that consists of 145 cities, a geo-stationary satellite, and three mobile satellites.

Liu *et al.* [37] proposed an immune system-based genetic algorithm for the DTSP where the topology of cities changes. A permutation-based dual operator is integrated into the immune operation to maintain diversity and a memory scheme is used to guide individuals to the promising areas of the search space. Dazhi and Shixin [11] proposed an agent-based evolutionary search to address a DTSP where cities are swapped. A recombination and local updating procedure is applied to the closest neighbor of each agent. The resulting offspring replaces the agent if it has better fitness; otherwise, it is accepted probabilistically to enhance exploration.

Tinós *et al.* [56] integrated an explicit memory to an elitism-based immigrants genetic algorithm (EIGA) to address both DTSPs where the topology of cities changes and where the links of the cities vary cyclically. The best individuals are stored in the memory and they are retrieved via the recombination operator (i.e., parents are selected from both the actual population and the memory).

Simões and Costa [51] integrated immigrants schemes with cross-generation elitism selection, heterogeneous recombination and cataclysmic mutation (CHC)-based algorithm for the DTSP with weight changes. The CHC framework does not use a normal mutation but reinitializes the population by preserving only the best individual and randomly swaps a part of it to generate the remaining individuals.

C. Other Methods

A discrete version of particle swarm optimization was applied to the DTSP where the topology changes [4], [5]. A pheromone table was used to allow particles to communicate. In [46], a scatter search algorithm was hybridized with a prediction mechanism for diversity maintenance whereas in [34] a simple parallel multistart of the common 2-Opt [9] local search operator was investigated for the DTSP.

IV. Proposed ACO-Based Memetic Algorithm

A. Memetic Framework

Local search algorithms can better explore locally a neighborhood in the search space. For our case, the neighborhood of a solution s is a set of different solutions that can be generated from s with a single move. For example, in the popular 2-Opt local search algorithm two arcs are deleted and reconnected in a different possible way. This specific move generates a neighbor solution. The same holds for the 3-Opt local search algorithm where three arcs are reconnected to generate a neighbor solution.

Many of the described algorithms from the literature (Section III) are not integrated with a local search operator to further improve the solution quality [16], [23], [37], [41], [51]. On the other hand, the algorithms that integrate local search improvements apply the operator either to the best solution [39] or to all solutions [34], [44], [46], [56].

Recently, $\mathcal{MM}\text{AS}_{\text{US}}$ was proposed for symmetric DTSPs [38]. The US operator may not be a very popular local search operator, but it was previously used to solve difficult combinatorial optimization problems under static environments with promising results [17], [18]. $\mathcal{MM}\text{AS}_{\text{US}}$ clearly showed that a local search operator can significantly improve the solution quality of a conventional $\mathcal{MM}$AS.[2] The experiments showed that $\mathcal{MM}$AS can provide promising starting points for the US operator for symmetric DTSP cases. In this paper, the US operator is extended to address both symmetric and asymmetric DTSP cases (see more details in Section IV-D).

[2]Thomas Stützle's ACOTSP, Version 1.03. Available at http://www.aco-metaheuristic.org/aco-code.

Algorithm 1 $\mathcal{MM}\text{AS}_{\text{US}}$

INPUT: none
OUTPUT: s^{bs} % best solution at any time
$t \leftarrow 0$ % iteration count
s^{ib} % iteration best ant
τ_0 % initial pheromone trail
τ_{min}, τ_{max} % minimum and maximum pheromone trail
InitializePheromones(τ_0)
while (termination condition *not* satisfied) **do**
  ConstructSolutions % using Eq. (5)
  $s^{ib} \leftarrow$ FindBest
  if $(f(s^{ib}, t) < f(s^{bs}, t))$ **then**
    $s^{bs} \leftarrow s^{ib}$
    UpdatePheromoneBounds(τ_{min}, τ_{max})
    *US*LocalSearch(s^{bs}) % using Algorithm 2
  end if
  UpdatePheromones % using Eqs. (6) and (7)
  if (stagnation behaviour *detected*) **then**
    InitializePheromones(τ_{max})
  end if
  $t \leftarrow t + 1$
end while

Stützle and Hoos [53] applied local search operators to the iteration-best ant of $\mathcal{MM}$AS after every iteration, whereas in [54], they further applied local search operators to all ants. Such extensive usage of local search may not be very efficient for dynamic optimization problems (DOPs) because the computation time naturally increases significantly since local search operators are computationally expensive algorithms. As discussed before, for DOPs, algorithms must produce high quality solutions quickly [56]. Therefore, the local search operator in $\mathcal{MM}\text{AS}_{\text{US}}$ is applied to the best-so-far-ant (a special ant that may not necessarily belong to the current population) only when a new best solution is found. This is because the local search operator is executed until no further improvement is possible. In case a new best solution is not found, the local search is not applied because it will unnecessarily increase the computation time to potentially "improve" a solution for which basically no further improvement is possible.

The general framework of the ACO-based memetic algorithm $\mathcal{MM}\text{AS}_{\text{US}}$ is given in Algorithm 1.

B. Constructing Solutions

Ants read pheromones to construct solutions and write pheromones to store solutions. Each ant k uses a probabilistic rule to choose the next city to visit. The probability of the kth ant to move from city v_i to city v_j is calculated as follows:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \text{ if } j \in \mathcal{N}_i^k \tag{5}$$

where τ_{ij} and η_{ij} are the existing pheromone trail and the heuristic information available *a priori* between cities v_i and v_j, respectively. The heuristic information is defined as $\eta_{ij} = 1/d'_{ij}$ where d'_{ij} is defined as in Eq. (3). $\mathcal{N}_i^k$ is the set of unvisited cities for ant k adjacent to city v_i. α and β are the two parameters which determine the relative influence of τ_{ij} and η_{ij}, respectively.

C. Pheromone Update Policy

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 evaporation, the best ant deposits pheromone as follows:

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

where $\Delta\tau_{ij}^{\text{best}} = 1/C^{\text{best}}$ is the amount of pheromone that the best ant deposits and C^{best} defines the solution quality of tour s^{best}. The best ant that is allowed to deposit pheromone may be either the best-so-far ant (s^{bs}), in which case $C^{\text{best}} = C^{\text{bs}}$, or the iteration-best ant (s^{ib}), in which case $C^{\text{best}} = C^{\text{ib}}$, where C^{bs} and C^{ib} define the solution quality of the best-so-far ant and the iteration-best ant, respectively. These two types of ants are applied in an alternate way. More precisely, the iteration-best ant is allowed to deposit pheromone to early iterations and the emphasis is gradually shifted to the best-so-far ant (more details in [54]). In this way, a stronger exploration is achieved at early stages of the optimization process and a stronger exploitation at later stages.

The lower and upper limits $\tau_{\min}$ and $\tau_{\max}$ of the pheromone trail values are imposed. The $\tau_{\max}$ value is bounded by $1/(\rho C^{\text{bs}})$, where C^{bs} is initially the solution quality of an estimated optimal tour (calculated by a nearest-neighbor heuristic) and later on is updated whenever a new best-so-far ant solution quality is found. The $\tau_{\min}$ value is set to $\tau_{\min} = \tau_{\max}/2n$.

Since a local search operator generates strong exploitation, the algorithm may lose its adaptation capabilities. To maintain the diversity, the pheromone trails are occasionally reinitialized to the value $\tau_{\max}$. For example, whenever the stagnation behavior[3] occurs or when no improved solution is found for a given number of iterations, the pheromone trails are reinitialized. This is a mechanism embedded to the conventional $\mathcal{MM}$AS algorithm and it is very useful for $\mathcal{MM}\text{AS}_{\text{US}}$, especially after US is applied.

D. US Local Search Operator

The US operator basically removes (or "unstrings") and inserts (or "strings") nodes from a tour into such position that improves the overall tour cost. In [38], only symmetric cases of DTSP were considered and tackled with types I and II insertions (see Figs. 1 and 2) and types I and II removals (see Figs. 5 and 6), whereas in this paper both symmetric and asymmetric DTSPs are considered. Hence, it is necessary to extend the US operator to cope with the asymmetric cases. Two other types of insertions and removals are defined for the DTSP: types III and IV insertions (see Figs. 3 and 4) and

[3]Detected using the λ-branching scheme [19] that calculates the statistics regarding the distribution of the current pheromone trails.

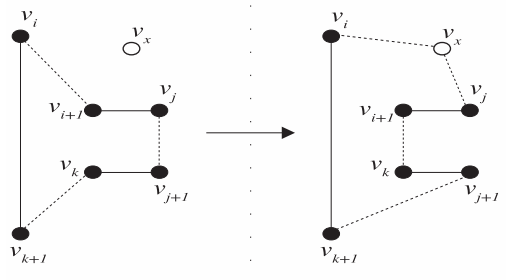

Fig. 1. Type I insertion of v_x between v_i and v_j.

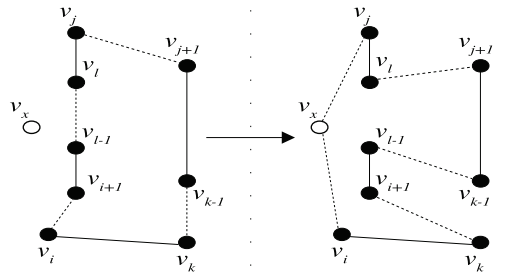

Fig. 2. Type II insertion of v_x between v_i and v_j.

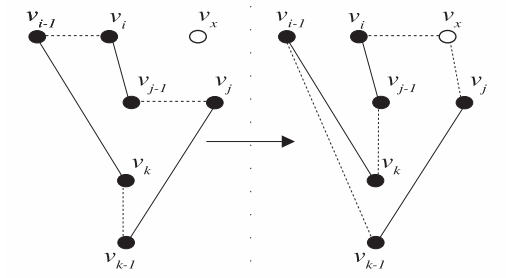

Fig. 3. Type III insertion of v_x between v_i and v_j.

types III and IV removals (see Figs. 7 and 8) to effectively tackle asymmetric cases.

Suppose that node v_x needs to be inserted between any two nodes v_i and v_j. The main feature of the insertion procedure of US is that when a node v_x is inserted, it is not necessarily placed between two consecutive nodes. However, after the insertion, these two nodes become adjacent to v_x. For a given orientation of a tour, we consider a node v_k in the subtour from v_j to v_i and a node v_l in the subtour from v_i to v_j. For any node v_h on the tour, we also consider v_{h+1} its successor and v_{h-1} its predecessor. Since the potential number of choices for v_i, v_j, and v_x could be large, the search is restricted within a neighborhood of a given size l (suggested values $l \in [3, 5]$ [21]). This is implemented in most local search operators to improve the computation time [53]. Basically, the neighbors of a node v_j are the nearest (e.g., the minimum weight) successors and predecessors among all nodes already included in the tour. The selection of the best place to insert a node in the tour is now constrained to the neighborhood of each node involved in the alternative under consideration as well as the alternatives tested to the removal procedures.

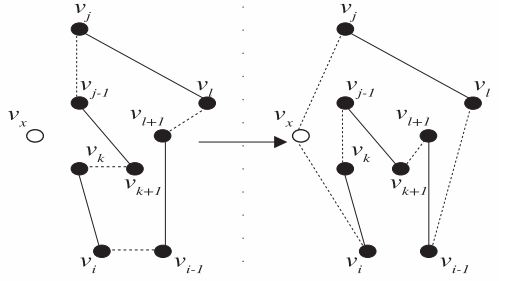

Fig. 4. Type IV insertion of v_x between v_i and v_j.

More precisely, the stringing procedure suggests four possible types of insertions of node v_x into the tour as shown in Figs. 1–4, respectively.

1) *Type I Insertion:* Assume that $v_k \neq v_i$ and $v_k \neq v_j$. The insertion of v_x results in the deletion of arcs (v_i, v_{i+1}), (v_j, v_{j+1}), and (v_k, v_{k+1}); and the insertion of arcs (v_i, v_x), (v_x, v_j), (v_{i+1}, v_k), and (v_{j+1}, v_{k+1}). Also, the subtours $(v_{i+1}, \ldots, v_j)$ and $(v_{j+1}, \ldots, v_k)$ are reversed.
2) *Type II Insertion:* Assume that $v_k \neq v_j$, $v_k \neq v_{j+1}$, $v_l \neq v_i$, and $v_l \neq v_{i+1}$. The insertion of v_x results in the deletion of arcs (v_i, v_{i+1}), (v_{l-1}, v_l), (v_j, v_{j+1}), and (v_{k-1}, v_k); and the insertion of arcs (v_i, v_x), (v_x, v_j), (v_l, v_{j+1}), (v_{k-1}, v_{l-1}), and (v_{i+1}, v_k). As above, the subtours $(v_{i+1}, \ldots, v_{l-1})$ and $(v_l, \ldots, v_j)$ are reversed.
3) *Type III Insertion:* Basically, this type of insertion can be seen as the inverse of type I insertion. When node v_x is inserted between v_i and v_j, the subtour of nodes is rearranged in such way that almost the whole sequence is inverted. The aim is to explore other promising regions of the search space. As in type I insertion, assume $v_k \neq v_i$ and $v_k \neq v_j$. The insertion of v_x results in the deletion of arcs (v_{i-1}, v_i), (v_{j-1}, v_j), and (v_{k-1}, v_k); and the insertion of arcs (v_i, v_x), (v_x, v_j), (v_k, v_{j-1}), and (v_{k-1}, v_{i-1}). As above, the subtours $(v_i, \ldots, v_{j-1})$ and $(v_k, \ldots, v_{i-1})$ are reversed.
4) *Type IV Insertion:* Analogously, this type of insertion can be seen as the reverse of type II insertion. As in type II, assume that $v_k \neq v_j$, $v_k \neq v_{j+1}$, $v_l \neq v_i$, and $v_l \neq v_{i+1}$. The insertion of v_x results in the deletion of arcs (v_{i-1}, v_i), (v_l, v_{l+1}), (v_{j-1}, v_j), and (v_k, v_{k+1}); and the insertion of arcs (v_i, v_x), (v_x, v_j), (v_{i-1}, v_l), (v_{l+1}, v_{k+1}), and (v_k, v_{j-1}). As above, the subtours $(v_i, \ldots, v_l)$ and $(v_{l+1}, \ldots, v_{j-1})$ are reversed.

The unstringing procedure is basically the opposite of the stringing procedure. Since there are four types of insertions there are four corresponding types of removals of node v_i, as shown in Figs. 5–8, respectively.

1) *Type I Removal:* Assume that v_j belongs to the neighborhood of v_{i+1} and v_k belongs to the neighborhood of v_{i-1}, with v_k being part of the subtour $(v_{i+1}, \ldots, v_{j-1})$. The removal of node v_i results in the deletion of arcs (v_{i-1}, v_i), (v_i, v_{i+1}), (v_k, v_{k+1}), and (v_j, v_{j+1}); and the insertion of arcs (v_{i-1}, v_k), (v_{i+1}, v_j), and (v_{k+1}, v_{j+1}). Also, the subtours $(v_{i+1}, \ldots, v_k)$ and $(v_{k+1}, \ldots, v_j)$ are reversed.

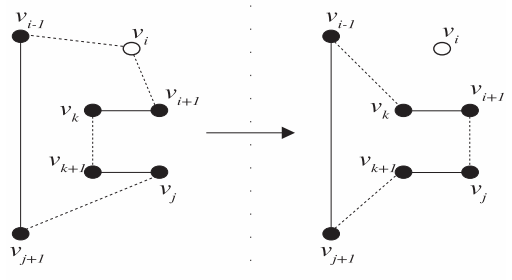

Fig. 5. Type I removal of v_i from the tour.

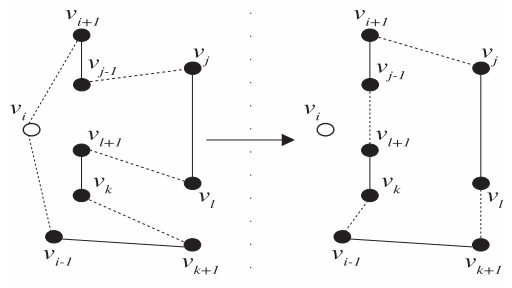

Fig. 6. Type II removal of v_i from the tour.

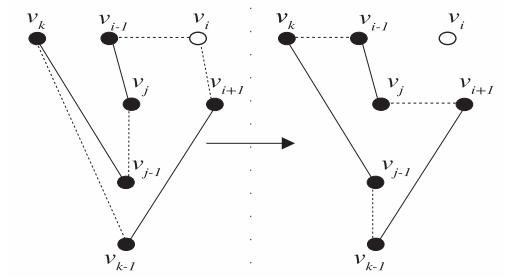

Fig. 7. Type III removal of v_i from the tour.

2) *Type II Removal:* Assume that v_j belongs to the neighborhood of v_{i+1}, v_k belongs to the neighborhood of v_{i-1}, with v_k being part of the subtour $(v_{j+1}, \ldots, v_{i-2})$ and v_l belongs to the neighborhood of v_{k+1}, with v_l being part of the subtour $(v_j, \ldots, v_{k-1})$. The removal of node v_i results in the deletion of arcs (v_{i-1}, v_i), (v_i, v_{i+1}), (v_{j-1}, v_j), (v_k, v_{k+1}), and (v_l, v_{l+1}); and the insertion of arcs (v_{i-1}, v_k), (v_{l+1}, v_{j-1}), (v_{i+1}, v_j), and (v_l, v_{k+1}). As above, the subtours $(v_{i+1}, \ldots, v_{j-1})$ and $(v_{l+1}, \ldots, v_k)$ are reversed.
3) *Type III Removal:* Assume that v_j belongs to the neighborhood of v_{i+1} and v_k belongs to the neighborhood of v_{i-1} with v_k being part of the subtour $(v_{i+1}, \ldots, v_{j-1})$. The removal of node v_i results in the deletion of arcs (v_{i-1}, v_i), (v_i, v_{i+1}), (v_{j-1}, v_j), and (v_{k-1}, v_k); and the insertion of arcs (v_{i+1}, v_j), (v_{i-1}, v_k), and (v_{j-1}, v_{k-1}). As above, the subtours $(v_{j-1}, \ldots, v_k)$ and $(v_{i-1}, \ldots, v_j)$ are reversed.
4) *Type IV Removal:* Assume that v_j belongs to the neighborhood of v_{i+1}, v_k belongs to the neighborhood of v_{i-1} with v_k being part of the subtour $(v_{l+1}, \ldots, v_{i-2})$, and v_l belongs to the neighborhood of v_{j-1} with v_l being part of the subtour $(v_{j+1}, \ldots, v_{k-1})$. The removal of node v_i results in the deletion of arcs (v_{i-1}, v_i), (v_i, v_{i+1}), (v_{j-1}, v_j), (v_{l-1}, v_l), and (v_k, v_{k+1}); and the insertion of arcs (v_k, v_{i-1}), (v_{j-1}, v_l), (v_{k+1}, v_{l-1}), and (v_j, v_{i+1}). As above, the subtours $(v_{k+1}, \ldots, v_{i-1})$ and $(v_j, \ldots, v_{l-1})$ are reversed.

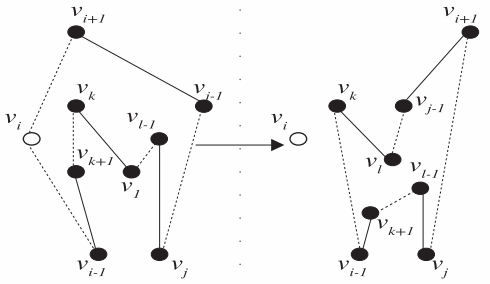

Fig. 8. Type IV removal of v_i from the tour.

Algorithm 2 US LocalSearch(s^{bs})

```
INPUT: s^bs      % best solution from MMAS
OUTPUT: none
while improvement do
  for (i = 1 to n) do
    CalculateRemovals(s^bs[i]) % all types
    ApplyBestRemoval(s^bs)     % least change
    CalculateInsertions(s^bs[i]) % all types
    ApplyBestInsertion(s^bs)   % greatest improvement
  end for
end while
```

One key feature of the US operator is that it may allow controlled moves that do not improve. For example, while the operator is executed, the current move may not improve the current solution but it is allowed to temporarily degrade the quality slightly to explore some unvisited (possibly promising) areas in the search space when executing the local search. This feature allows the operator to deal with possible traps of poor local optima. In addition, searching to the same areas is avoided because these controlled moves that degrade the solution quality are performed while testing different types of insertions and removals, but only the ones that improve are finally selected.

The overall execution of the US local search operator is given in Algorithm 2. More precisely, the CalculateRemovals method calculates the changes to the cost of the best solution s^{bs} for all types of removals and applies the removal move that causes the least change using the ApplyBestRemoval method. Then, the CalculateInsertions method calculates the changes to the cost of the best solution s^{bs} for all types of insertions and applies the insertion move that causes the greatest improvement using the ApplyBestInsertion method. The neighborhood is evaluated with the addition of the difference caused from the moves of the US operator and the cost of the original best solution s^{bs}. The execution terminates when no improvement is discovered for all nodes; otherwise, the process will be restarted from the resulting solution.

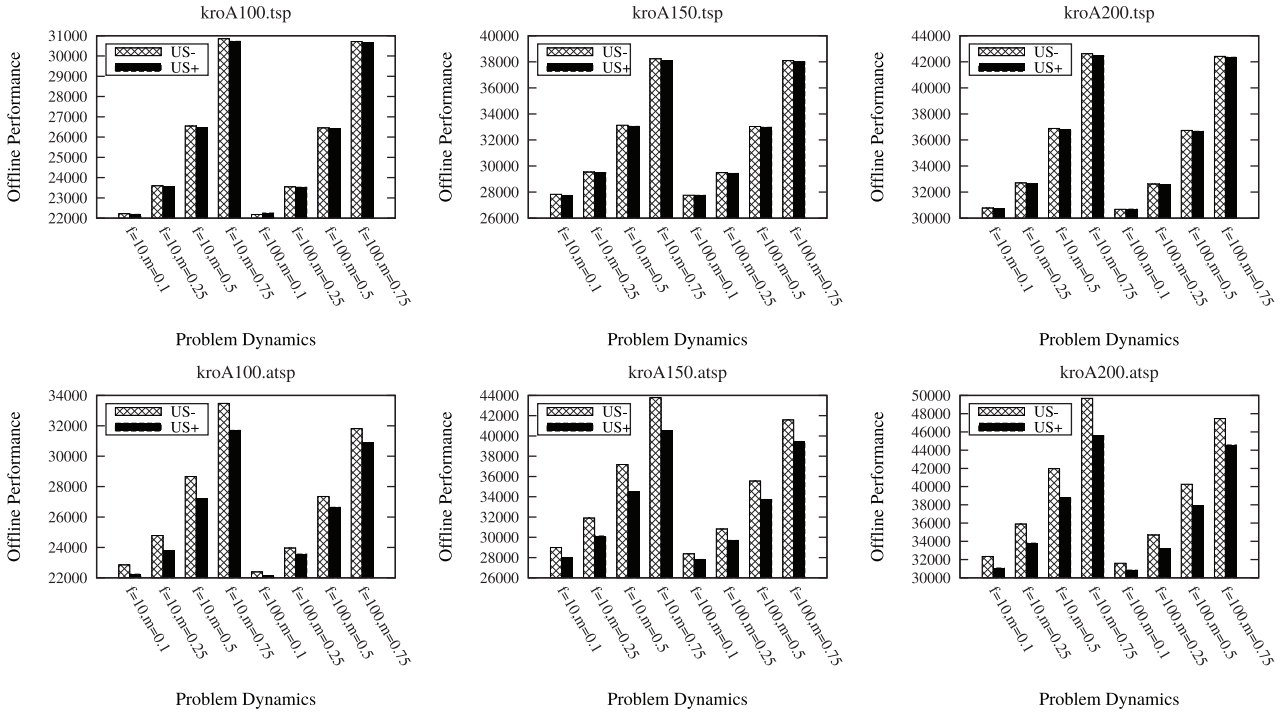

Fig. 9. Offline performance of $\mathcal{MM}\text{AS}_{\text{US}}$ without (US−) and with (US+) type III/IV moves, respectively, for symmetric (top) and asymmetric (bottom) DTSPs.

The time complexity of a naive US operator (without taking into account any speed up techniques) is determined by the $\mathcal{O}(n^4)$ choices of v_i, v_j, v_l, and v_k nodes for each of the CalculateRemovals and CalculateInsertions methods. For ApplyBestRemoval and ApplyBestInsertion methods, it requires $\mathcal{O}(n)$ time for each one, to update the weights for the fact that v_i is removed from the tour and that v_x is added in the tour, respectively, resulting in $\mathcal{O}(n^4 + n) = \mathcal{O}(n^4)$. Since the procedures are performed n times (i.e., for all nodes), the overall time complexity of the US operator is $\mathcal{O}(n^5)$.

V. Experimental Study

A. Experimental Setup

1) Generating Dynamic Environments: The benchmark generator described in Section II-C can convert any static TSP problem instance into a DTSP. Three (static) TSP benchmark instances were obtained from TSPLIB, i.e., `kroA100.tsp`, `kroA150.tsp`, and `kroA200.tsp`, to generate different test cases of DTSPs. More precisely, the frequency of change was set to $f = 10$ and $f = 100$ iterations indicating environmental changes of high and low frequencies, respectively, and the magnitude of change was set to $m = 0.1$, $m = 0.25$, $m = 0.5$, and $m = 0.75$, indicating the degree of environmental changes from small, to medium, and to large, respectively. As a result, eight DTSPs (i.e., two values of $f \times$ four values of m) with lower and upper bounds $F_L = 0$ and $F_U = 2$, respectively, were generated from each problem instance, both symmetric and asymmetric, to systematically analyze the proposed memetic algorithm. For each DTSP test case, 100 environmental changes were allowed. All asymmetric problem instances have an extension of `.atsp` in the problem label.

2) Performance Measurements: An observation of the best-so-far solution after a dynamic change was recorded every iteration and used to evaluate the performance for 30 independent executions (with a different random seed for an algorithm and the same random seed for the dynamic environment on each execution). Therefore, the overall offline performance [27] is defined as follows:

$$\bar{P}_{\text{OFF}} = \frac{1}{I}\sum_{i=1}^{I}\left(\frac{1}{E}\sum_{j=1}^{E} P_{ij}^{*}\right) \tag{8}$$

where I is the total number of iterations, E is the number of independent executions, and P_{ij}^{*} is the best-so-far solution cost (after a change) of iteration i of execution j. For a fair comparison, all the algorithms performed the same number of iterations, and for each iteration the same number of ants (evaluations) is allowed. The experiments were performed under Linux Systems with an Intel Core i7-3930K 3.20 GHz processor with 12 MB cache and 16 GB RAM. The CPU times (in seconds) of the environmental changes were recorded and averaged among all executions (i.e., E runs). Note that, since we are dealing with heuristic methods, the CPU time of the algorithms may not be exactly the same for each environmental change. Although this fact may degrade the fairness of the comparisons a bit, the CPU times are not significantly different according to our experimental results (see Table II later on).

3) Parameter Settings: The common parameters for all ACO algorithms used were set to typical values, i.e., $\alpha = 1$ and $\beta = 5$ in Eq. (5) for all the experiments. The colony size consists of 50 ants for all algorithms. Since the parameter ρ in Eq. (6) is an important parameter for ACO algorithms when addressing dynamic environments [40]; we have performed

TABLE I
EXPERIMENTAL RESULTS REGARDING THE SOLUTION QUALITY (OFFLINE PERFORMANCE) OF $\mathcal{MM}$AS$_{\text{US}}$ AGAINST OTHER MEMETIC ALGORITHMS FOR DIFFERENT DTSPS, WHERE BOLD VALUES INDICATE STATISTICAL SIGNIFICANCE

Symmetric Travelling Salesman Problem												
Algorithms & DTSPs	kroA100.tsp				kroA150.tsp				kroA200.tsp			
$f = 10$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
$\mathcal{MM}$AS$_{2-Opt}$	22576	24267	27708	32507	28441	30576	34829	40710	31397	33929	38991	45704
$\mathcal{MM}$AS$_{3-Opt}$	22235	23660	26687	31055	27856	29646	33334	38508	30812	32801	37086	42946
$\mathcal{MM}$AS$_{res-3-Opt}$	22434	24179	27756	32542	28246	30521	34971	40833	31157	33836	39199	45913
$\mathcal{MM}$AS$_{US}$	**22186**	**23552**	**26465**	**30726**	**27744**	**29494**	**33045**	**38076**	**30722**	**32635**	**36772**	**42451**
$f = 100$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
$\mathcal{MM}$AS$_{2-Opt}$	22343	23865	27013	31491	28079	30054	34001	39494	30987	33268	38106	44446
$\mathcal{MM}$AS$_{3-Opt}$	22197	23602	26587	30865	27809	29557	33193	38333	30752	32703	36949	42773
$\mathcal{MM}$AS$_{res-3-Opt}$	22251	23749	26874	31337	27907	29792	33762	39268	30810	33026	37804	44123
$\mathcal{MM}$AS$_{US}$	**22162**	**23523**	**26405**	**30646**	**27754**	**29432**	**32970**	**37994**	**30686**	**32575**	**36667**	**42345**
Asymmetric Travelling Salesman Problem												
Algorithms & DTSPs	kroA100.atsp				kroA150.atsp				kroA200.atsp			
$f = 10$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
$\mathcal{MM}$AS$_{2-Opt}$	24010	26571	30723	35714	31059	34547	39695	46404	35404	39298	45236	52872
$\mathcal{MM}$AS$_{3-Opt}$	22606	24963	30022	35398	28436	31790	38810	46091	31575	35549	43917	52297
$\mathcal{MM}$AS$_{res-3-Opt}$	22670	24380	27850	32386	28614	30777	34918	40807	31873	34351	39096	45835
$\mathcal{MM}$AS$_{US}$	**22241**	**23781**	**27212**	**31701**	**27998**	**30132**	**34509**	**40529**	**31071**	**33803**	**38807**	**45597**
$f = 100$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
$\mathcal{MM}$AS$_{2-Opt}$	22697	24500	27942	32425	28759	31520	36120	42230	32296	35997	41426	48544
$\mathcal{MM}$AS$_{3-Opt}$	22321	23991	27718	32291	28013	30592	36002	42127	31131	34406	41428	48458
$\mathcal{MM}$AS$_{res-3-Opt}$	22388	23883	27129	31457	28163	30272	34318	40129	31297	33829	38579	45267
$\mathcal{MM}$AS$_{US}$	**22145**	**23565**	**26641**	**30881**	**27820**	**29696**	**33733**	**39463**	**30841**	**33215**	**37890**	**44562**

parameter tuning and the evaporation rate was set to $\rho = 0.8$. Further details for tuning the evaporation rate parameter can be found in the supplementary document of this paper.

B. Experiments on the Effect of Types III and IV Moves

To investigate the effect of types III and IV moves on the performance of $\mathcal{MM}$AS$_{\text{US}}$, an algorithm variation where types III and IV moves are allowed is compared with another algorithm variation where types III and IV moves are omitted. Fig. 9 presents the offline performance of $\mathcal{MM}$AS$_{\text{US}}$ without types III and IV insertions/removals, denoted US$-$,[4] and with types III and IV insertions/removals, denoted US$+$, for both symmetric and asymmetric DTSPs.

From Fig. 9, it can be observed that the performance of the two variations is almost identical on symmetric cases (see top figures) whereas it is significantly different on asymmetric cases (see bottom figures). Specifically, when types III and IV moves are allowed on asymmetric cases, they have a greater impact than when allowed on symmetric cases. This shows the effectiveness of the asymmetric extension of the US operator designed to cope with the asymmetric cases.

C. Experiments With Other Local Search Algorithms

To evaluate the strengths and weaknesses of the proposed memetic algorithm, i.e., $\mathcal{MM}$AS$_{\text{US}}$, two of the most popular and commonly used local search algorithms are considered: 2-Opt [9] due to its efficiency and 3-Opt [36] due to its performance. These local search algorithms are integrated in exactly the same way as the US operator is applied to $\mathcal{MM}$AS$_{\text{US}}$ (i.e., applied only to the new best-so-far solution each time it is discovered until no further improvement is possible and the best possible move is considered) for fair comparison. Hence, their integrations are denoted as $\mathcal{MM}$AS$_{2-\text{Opt}}$ and $\mathcal{MM}$AS$_{3-\text{Opt}}$, respectively. Similarly, as with $\mathcal{MM}$AS$_{\text{US}}$, the near-neighbor lists and do not look bits techniques are used for $\mathcal{MM}$AS$_{2-\text{Opt}}$ and $\mathcal{MM}$AS$_{3-\text{Opt}}$ [3]. Similarly with the US operator in Algorithm 2, there is no need to re-evaluate an entire solution for 2-Opt and 3-Opt since the difference caused by the exchanges can be added to the original solution.

Note that the proposed $\mathcal{MM}$AS$_{\text{US}}$ is suitable for both symmetric and asymmetric cases, whereas $\mathcal{MM}$AS$_{2-\text{Opt}}$ and $\mathcal{MM}$AS$_{3-\text{Opt}}$ are only suitable for symmetric cases. A straightforward way to address this issue is to transform an asymmetric case to a symmetric one and apply them as mentioned previously. However, it may be a convenient choice for static problems that require a transformation once. For dynamic problems, it may require more time to perform/maintain the transformation, e.g., whenever a change occurs, and hence increases the computation time significantly (especially in large problems). Besides that, the effectiveness of the methods may be degraded [29], [52]. Therefore, for the sake of comparison, $\mathcal{MM}$AS$_{2-\text{Opt}}$ and $\mathcal{MM}$AS$_{3-\text{Opt}}$ are modified (e.g., subtour reversals are considered). Literally there are no any "2-opt" or "3-opt" moves for asymmetric cases. The subtour reversal results in several node exchanges depending on the length of the subtour and diversifies the original solution more than what a normal local search is supposed to. But, there is one special 3-opt move that subtour reversals are not performed and the direction of the tour is preserved, which is commonly used for asymmetric cases. This restricted "asymmetric" version of 3-Opt, denoted res-3-Opt [53], is also integrated with $\mathcal{MM}$AS, denoted $\mathcal{MM}$AS$_{\text{res}-3-\text{Opt}}$ and used in our comparisons.

The experimental results regarding the offline performance of the memetic $\mathcal{MM}$AS for all DTSPs are

[4]Similar with the previous version of the algorithm presented in [38].

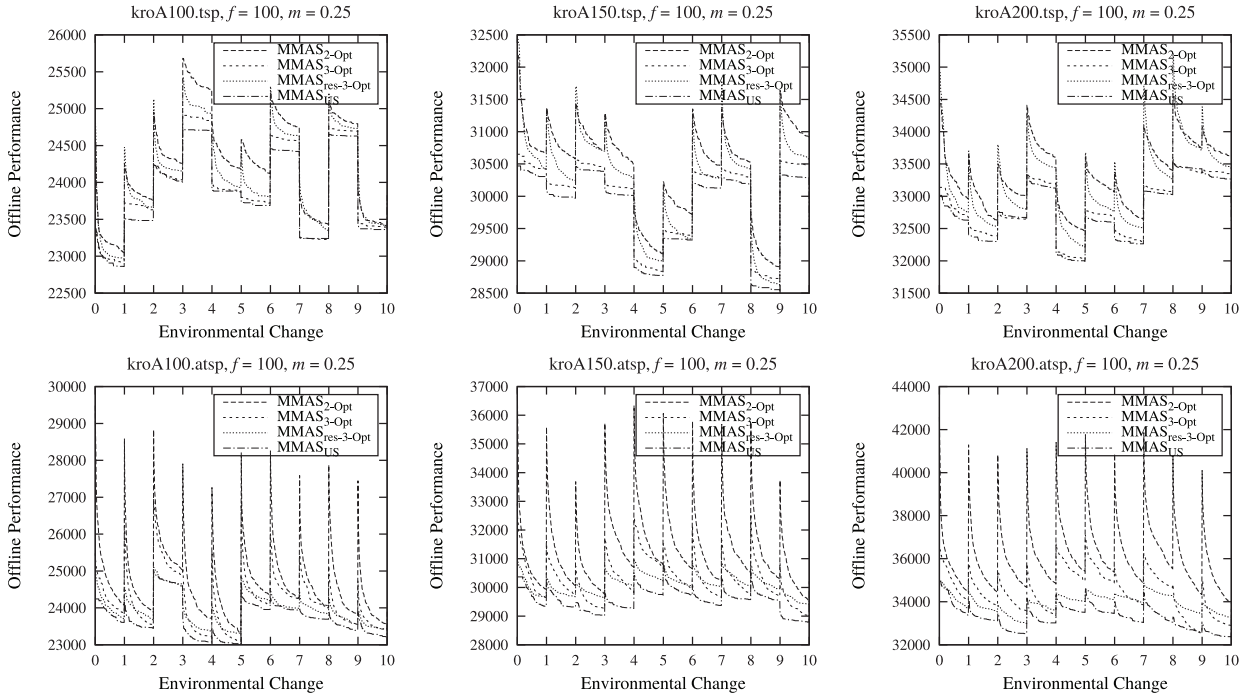

Fig. 10. Dynamic behavior of memetic algorithms for symmetric (top) and asymmetric (bottom) DTSPs.

TABLE II
EXPERIMENTAL RESULTS REGARDING THE COMPUTATION TIME (CPU-TIME IN SECONDS) REQUIRED BY $\mathcal{MM}\text{AS}_{\text{US}}$ TO OUTPUT A SOLUTION FOR EACH ENVIRONMENTAL CHANGE AGAINST OTHER MEMETIC ALGORITHMS FOR DIFFERENT DTSPS

Symmetric Travelling Salesman Problem												
Algorithms & DTSPs	kroA100.tsp				kroA150.tsp				kroA200.tsp			
$f = 10,\ m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
$\mathcal{MM}\text{AS}_{2-Opt}$	0.02	0.02	0.02	0.02	0.05	0.05	0.05	0.05	0.08	0.08	0.08	0.08
$\mathcal{MM}\text{AS}_{3-Opt}$	0.02	0.02	0.02	0.02	0.05	0.05	0.05	0.05	0.08	0.08	0.08	0.08
$\mathcal{MM}\text{AS}_{res-3-Opt}$	0.02	0.02	0.02	0.02	0.05	0.05	0.05	0.05	0.08	0.08	0.08	0.08
$\mathcal{MM}\text{AS}_{US}$	0.03	0.03	0.03	0.03	0.06	0.06	0.06	0.06	0.10	0.10	0.11	0.11
$f = 100,\ m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
$\mathcal{MM}\text{AS}_{2-Opt}$	0.24	0.25	0.25	0.26	0.49	0.49	0.49	0.50	0.83	0.83	0.86	0.83
$\mathcal{MM}\text{AS}_{3-Opt}$	0.25	0.25	0.24	0.25	0.49	0.51	0.49	0.49	0.82	0.82	0.83	0.83
$\mathcal{MM}\text{AS}_{res-3-Opt}$	0.24	0.25	0.25	0.25	0.49	0.50	0.49	0.52	0.83	0.83	0.83	0.83
$\mathcal{MM}\text{AS}_{US}$	0.25	0.25	0.25	0.25	0.51	0.51	0.51	0.51	0.84	0.85	0.86	0.86
Asymmetric Travelling Salesman Problem												
Algorithms & DTSPs	kroA100.atsp				kroA150.atsp				kroA200.atsp			
$f = 10,\ m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
$\mathcal{MM}\text{AS}_{2-Opt}$	0.03	0.03	0.03	0.03	0.06	0.06	0.06	0.06	0.10	0.10	0.10	0.10
$\mathcal{MM}\text{AS}_{3-Opt}$	0.02	0.03	0.03	0.03	0.06	0.06	0.06	0.06	0.10	0.10	0.10	0.10
$\mathcal{MM}\text{AS}_{res-3-Opt}$	0.05	0.05	0.05	0.05	0.09	0.10	0.10	0.10	0.15	0.15	0.15	0.16
$\mathcal{MM}\text{AS}_{US}$	0.04	0.04	0.05	0.05	0.08	0.11	0.12	0.13	0.16	0.19	0.19	0.20
$f = 100,\ m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
$\mathcal{MM}\text{AS}_{2-Opt}$	0.54	0.55	0.56	0.56	0.95	0.97	0.99	1.00	1.47	1.52	1.54	1.56
$\mathcal{MM}\text{AS}_{3-Opt}$	0.55	0.55	0.56	0.57	0.94	0.97	1.00	1.01	1.45	1.49	1.56	1.55
$\mathcal{MM}\text{AS}_{res-3-Opt}$	0.54	0.55	0.56	0.56	0.95	0.97	0.98	1.00	1.45	1.49	1.52	1.53
$\mathcal{MM}\text{AS}_{US}$	0.55	0.56	0.58	0.59	0.97	1.00	1.03	1.04	1.49	1.54	1.56	1.59

presented in Table I. Statistical tests are performed as follows: Kruskal–Wallis for multiple comparisons, followed by posthoc paired comparisons using Mann–Whitney tests with the Bonferroni correction. In Fig. 10, the dynamic offline performance for slowly changing environments for the first 10 dynamic changes of $\mathcal{MM}\text{AS}_{2-\text{Opt}}$, $\mathcal{MM}\text{AS}_{3-\text{Opt}}$, $\mathcal{MM}\text{AS}_{\text{res}-3-\text{Opt}}$ and $\mathcal{MM}\text{AS}_{\text{US}}$ are plotted to better understand the behavior of the ACO-based memetic algorithms in symmetric and asymmetric DTSPs. Table II presents the CPU-time results required by the investigated algorithms to output a solution for each environmental change for all DTSPs. Note that since the CPU-time of the algorithms is affected by: 1) how fast the local search operator terminates and 2) how often the local search is triggered (depending on the discovery of a new best solution as presented in Algorithm 1), the results presented are approximations (e.g., an average among the 100 dynamic changes). From the experimental results, the following observations can be drawn.

1) Comparisons Regarding the Offline Performance: In terms of solution quality, algorithms designed for symmetric cases, e.g., $\mathcal{MM}\text{AS}_{3-\text{Opt}}$, outperform algorithms designed for asymmetric cases, e.g., $\mathcal{MM}\text{AS}_{\text{res}-3-\text{Opt}}$, in most symmetric

DTSPs, and vice versa, for most asymmetric DTSPs. However, $\mathcal{MM}\text{AS}_{\text{US}}$ performs significantly better than $\mathcal{MM}\text{AS}_{2-\text{Opt}}$, $\mathcal{MM}\text{AS}_{3-\text{Opt}}$, and $\mathcal{MM}\text{AS}_{\text{res}-3-\text{Opt}}$ on all DTSPs, both symmetric and asymmetric (see Table I). From Fig. 10, we can observe that $\mathcal{MM}\text{AS}_{\text{US}}$ has strong exploitation and quickly contributes to the offline performance with better solutions. Although $\mathcal{MM}\text{AS}_{3-\text{Opt}}$ also shows strong exploitation, it gets stuck into a worse solution than the one of $\mathcal{MM}\text{AS}_{\text{US}}$, but it outperforms other algorithms.

If we observe the US operator closely; applying the right choice for the cities and neighborhoods embodies restricted k–Opt moves (with $k \geq 3$). Even though the local search operators have similar characteristics, the US operator does not stop early in local optima as the other type of operators. As a result, the (local) exploration in the search space is enhanced since it allows the predecessors and successors of the nodes to control the moves. The exploration leads to a better solution quality, which can be observed from Fig. 10, where $\mathcal{MM}\text{AS}_{\text{US}}$ maintains better offline performance for all the executing time. In fact, these (symmetric) results match our previous ones found in [38], where the symmetric variation of the $\mathcal{MM}\text{AS}_{\text{US}}$ operator was applied to the DTSP where cities are swapped. This supports the generality of the performance of our algorithm since it is not dependent on the type of dynamic changes for symmetric DTSPs.

2) Comparisons Regarding the Computational Time: In terms of the computation time, all algorithms require more computation time for asymmetric than symmetric DTSPs, which is natural because both directions are considered in asymmetric DTSPs. Specifically, $\mathcal{MM}\text{AS}_{\text{US}}$ is competitive with the competing algorithms on most symmetric and asymmetric cases although it is not the most efficient one in all DTSPs. On many asymmetric cases with $f = 10$, both $\mathcal{MM}\text{AS}_{\text{US}}$ and $\mathcal{MM}\text{AS}_{\text{res}-3-\text{Opt}}$ require more computation time than the other algorithms. This is because they are designed to cope with asymmetric cases and several improvements are more likely to be found, whereas $\mathcal{MM}\text{AS}_{2-\text{Opt}}$ and $\mathcal{MM}\text{AS}_{3-\text{Opt}}$ break quicker because further improvements are unlikely to be found.

Although $\mathcal{MM}\text{AS}_{\text{US}}$ is slightly computationally expensive in some test cases with $f = 10$ (both symmetric and asymmetric), the significant improvement on the solution quality from Table I shows it is worth the effort. With the exception of some asymmetric cases for `kroA150.atsp` and `kroA200.atsp`, for all the investigated algorithms, the estimated CPU time required to output their best solution is less than a second.

D. Experiments in Real-World Scenarios

For experimental reasons, the magnitude of change was fixed in previous experiments and the frequency of change was synchronized with the algorithmic iterations. In real-world scenarios, the magnitude of change is unknown and the frequency of change is synchronized with real-time. Therefore, further experiments are performed with more realistic scenarios generated with randomly chosen values of m from a uniform distribution in $[0, 0.25]$, $[0, 0.5]$, and $[0, 1]$, where the environments are more likely to have small, medium and large changes, respectively. The frequency of change was set to change every 15 and 30 s that indicates the available time an algorithm has to optimize on every environmental change. Both available time values are enough for the algorithms to recover when a dynamic change occurs for small problem instances. However, as the problem size increases, the algorithms naturally need more time (in terms of algorithmic iterations). Basically, the frequency of change increases as the problem size increases because the algorithms will have less available time to optimize (i.e., less algorithmic iterations will be executed). Only asymmetric dynamic changes are considered because most real applications have this characteristic. Therefore, this problem has a more practical value as described previously. In total, eight different problem instances are obtained from TSPLIB ranging from 51 to 1173 nodes. Ten environmental changes are allowed for the experiments. Moreover, in real-world situations, the offline performance described in Eq. (8) may not be suitable because only the best output provided for each environment counts. Hence, the best solution output from each environment was recorded and averaged over 30 independent runs.

$\mathcal{MM}\text{AS}_{\text{US}}$ is compared with the best performing heuristic for asymmetric TSPs, i.e., the Kanellakis–Papadimitriou (KP) [30] heuristic[5] and other state-of-the-art algorithms from the literature (see Section III): 1) P-ACO [24], an ACO framework developed to tackle DTSPs; 2) EIACO [41], one of the best performing ACO algorithms for DTSPs; 3) $\mathcal{MM}$AS [53], one of the state-of-the-art ACO algorithms in stationary environments; and 4) EIGA-GAPX [56], one of the best performing EAs[6] in evolutionary dynamic optimization. The experimental results regarding the mean among all best solutions outputs (i.e., for ten environments) of the aforementioned algorithms for the described scenario are presented in Table III. The same experimental settings and statistical tests are used as with the previous experiments.

When the scenarios change every 15 s, it can be observed from Table III that $\mathcal{MM}\text{AS}_{\text{US}}$ significantly outperforms the other algorithms in most problem instances with $m \in [0, 0.25]$ and $m \in [0, 0.5]$, with the exception of `berlin52.atsp`, `d198.atsp`, and `u574.atsp` (in some cases), which is comparable with the KP heuristic. When $m \in [0, 1]$, $\mathcal{MM}\text{AS}_{\text{US}}$ becomes also comparable with the KP heuristic on the two larger problem instances, e.g., `rat783.atsp` and `pcb1173.atsp`. This is probably because the resulting changing environments for this scenario may be completely different and the time available is extremely short for $\mathcal{MM}\text{AS}_{\text{US}}$ to recover for such cases.

The same observations occur when the scenarios change every 30 s in which more time is available for the algorithms to recover. The only difference from the results of the scenarios

[5]The KP heuristic performs primary sequential k-opt moves for some odd $k \geq 3$ followed by double-bridge moves (a special 4-opt move that preserves the direction of a tour) until no improving move of this type can be found. Since the original KP heuristic was developed for static TSP and is deterministic, an application of a restart approach of KP is used for DTSP in this paper to restart from a new solution when a local optimum is found and reoptimize from the best-so-far solution when a dynamic change occurs.

[6]The generalized asymmetric partitioned crossover (GAPX) is used, which performs better than other crossovers in asymmetric cases.

TABLE III
EXPERIMENTAL RESULTS REGARDING THE MEAN BEST OUTPUT OF $\mathcal{MM}$AS$_{\text{US}}$ WITH THE OTHER PEER ALGORITHMS ON DIFFERENT REAL-WORLD SCENARIOS, WHERE BOLD VALUES INDICATE STATISTICAL SIGNIFICANCE

Problem Instances	$\mathcal{MM}AS_{US}$	KP [30]	$\mathcal{MM}$AS [53]	P-ACO [24]	EIACO [41]	EIGA-GAPX [56]
m randomly generated $\in [0, 0.25]$ with $f = 15$ seconds						
berlin52.atsp	**7895.3**	**7865.3**	7923.2	8055.6	7916.0	8347.5
eil101.atsp	**652.5**	657.2	655.7	671.4	663.2	703.4
d198.atsp	**16680.4**	**16690.1**	16932.0	17679.6	17106.9	18203.2
lin318.atsp	**44787.0**	46019.9	45182.1	48588.1	46426.1	52737.9
pcb442.atsp	**53741.3**	55100.1	57226.9	60649.0	57460.5	65169.2
u574.atsp	**40255.7**	40570.7	44604.1	47231.7	43510.8	47487.0
rat783.atsp	**9534.6**	9727.1	10996.3	11609.0	10543.9	11587.4
pcb1173.atsp	**64412.2**	65764.8	77971.7	81969.6	72742.0	84265.1
m randomly generated $\in [0, 0.5]$ with $f = 15$ seconds						
berlin52.atsp	8285.8	**8259.3**	8313.2	8506.4	8327.6	8715.8
eil101.atsp	**697.8**	707.7	701.5	724.8	716.1	783.7
d198.atsp	**17594.9**	**17658.0**	17783.6	18707.5	18134.6	19588.6
lin318.atsp	**47383.9**	48569.2	47779.3	51670.3	48981.4	56478.4
pcb442.atsp	**59320.0**	60088.0	63858.8	67403.6	63569.9	72879.2
u574.atsp	**42976.2**	**42969.9**	47621.3	50416.9	46122.4	51026.5
rat783.atsp	**10187.8**	10311.2	11853.5	12447.3	11226.7	12826.6
pcb1173.atsp	**73097.2**	73564.4	88305.7	91737.9	81363.8	106143.2
m randomly generated $\in [0, 1]$ with $f = 15$ seconds						
berlin52.atsp	**8594.9**	8632.8	**8603.8**	8856.4	8634.5	9144.2
eil101.atsp	**750.8**	770.4	755.2	784.8	768.1	843.2
d198.atsp	18687.7	**18591.4**	18727.1	19774.5	19086.8	21127.5
lin318.atsp	**49843.6**	51711.4	50976.1	54641.3	51141.6	61448.2
pcb442.atsp	**66066.1**	66919.3	71487.4	75552.8	70377.6	80807.4
u574.atsp	**48487.9**	**48409.2**	53248.6	56169.7	51682.8	58439.9
rat783.atsp	**11971.9**	**12103.3**	13671.0	14303.1	12948.5	15957.5
pcb1173.atsp	**93686.7**	**93523.8**	110622.2	114821.3	102621.4	145865.0
m randomly generated $\in [0, 0.25]$ with $f = 30$ seconds						
berlin52.atsp	7897.3	**7863.7**	7911.4	8046.5	7919.7	8307.8
eil101.atsp	**650.7**	655.6	653.0	669.0	661.5	696.7
d198.atsp	**16629.5**	**16651.6**	16815.4	17477.0	17012.6	17855.0
lin318.atsp	**44396.8**	45905.8	44911.0	47616.6	46000.4	52337.2
pcb442.atsp	**53754.0**	54931.0	56211.6	58745.6	56936.4	64532.4
u574.atsp	**40062.6**	40464.1	43030.7	46386.8	43033.9	46946.1
rat783.atsp	**9487.3**	9695.4	10748.8	11370.1	10416.3	11453.9
pcb1173.atsp	**64259.4**	65489.3	76034.4	80172.1	71838.2	81796.5
m randomly generated $\in [0, 0.5]$ with $f = 30$ seconds						
berlin52.atsp	8265.2	**8237.3**	8299.2	8473.4	8298.5	8672.1
eil101.atsp	**696.9**	705.8	699.0	722.6	713.9	770.1
d198.atsp	**17572.5**	**17590.6**	17651.1	18586.6	18071.1	19414.7
lin318.atsp	**46923.8**	48439.6	**46993.1**	50694.0	48300.3	56334.9
pcb442.atsp	**59151.0**	59959.6	62465.5	65975.1	62877.6	72674.3
u574.atsp	**42788.1**	**42816.2**	46319.1	49472.7	45630.3	50736.3
rat783.atsp	**10167.6**	10297.3	11579.9	12206.9	11080.2	12426.6
pcb1173.atsp	**72766.4**	73327.9	86079.7	90127.3	80246.8	99820.1
m randomly generated $\in [0, 1]$ with $f = 30$ seconds						
berlin52.atsp	**8584.2**	8610.2	**8587.3**	8792.3	8627.4	8966.6
eil101.atsp	**749.1**	768.2	752.6	819.2	765.8	830.5
d198.atsp	**18468.7**	**18494.3**	18628.7	20196.5	19018.1	20625.8
lin318.atsp	**48869.7**	51466.6	49267.5	56111.2	50456.0	60855.7
pcb442.atsp	**65828.3**	66606.9	69335.8	76785.0	69731.3	80922.7
u574.atsp	**48445.2**	**48172.6**	51791.3	56641.7	51119.9	58121.8
rat783.atsp	**11914.3**	11993.0	13341.6	14207.5	12776.5	15061.3
pcb1173.atsp	**92916.7**	93473.5	108006.6	112048.4	101139.6	136732.2

that change every 15 s is that now $\mathcal{MM}$AS$_{\text{US}}$ significantly outperforms the KP heuristic when the scenario is $m \in [0, 1]$ on the two larger problem instances, which confirms our claim above. However, it remains comparable with the KP heuristic on the same instances that were exceptions previously. This observation shows that the performance of the algorithm may also depend on the environmental changes applied to the problem instance or the topology of the problem instance itself.

For the remaining competitors, $\mathcal{MM}$AS is competitive with $\mathcal{MM}$AS$_{\text{US}}$ and the KP heuristic on the four smaller problem instances for all scenarios. In contrast, the solution quality of P-ACO, EIACO, and EIGA-GAPX algorithms is far away from $\mathcal{MM}$AS$_{\text{US}}$ and KP for all problem instances for all scenarios.

VI. CONCLUSION

In this paper, we address the DTSP where the weights between cities change either symmetrically or asymmetrically. An ACO-based memetic algorithm, where $\mathcal{MM}$AS is integrated with the US local search operator, is designed to address both symmetric and asymmetric DTSPs. The US operator is enhanced with additional two more types of moves to cope especially with asymmetric cases without degrading its

performance on symmetric cases as other local search heuristics. The proposed $\mathcal{MM}\text{AS}_{\text{US}}$ algorithm tends to combine the adaptation capabilities of $\mathcal{MM}$AS and the solution quality improvement power of US.

From the experiments results of different studies, the following concluding remarks can be drawn. First, local search operators are essential tools for ACO when addressing DTSPs. They are capable of improving the solution quality significantly. However, the overuse of local search operators has to be avoided because the computational time will increase significantly. The proposed memetic framework maintains a good balance between the computation time and the solution quality of effective local search algorithms. Second, the US local search operator performs significantly better than other popular local search heuristics when integrated with $\mathcal{MM}$AS. $\mathcal{MM}\text{AS}_{\text{US}}$ performs equally well on both symmetric and asymmetric DTSPs when all (four) types of moves are used. Third, $\mathcal{MM}\text{AS}_{\text{US}}$ maintains high output efficiently even on quickly changing DTSPs. Such cases are more relevant from a practical point of view. Finally, the proposed memetic framework, where US is applied, is also effective for solving DTSPs even when other local search algorithms are used.

For the future work, the dynamic version of other routing problems that are basically variations [33] or direct extensions of the fundamental TSP, e.g., capacitated vehicle routing problems [48] and capacitated arc routing problems [49], [58], can be considered. For these problems, multiple routes are considered, which makes them closer to even more real-world applications in the important area of logistics. In fact, exact methods may not be appropriate at all for such dynamic problems because of the computational time required. Considering the efficiency (time) and performance (solution quality) of $\mathcal{MM}\text{AS}_{\text{US}}$, it will be a potential good candidate to tackle these problems with either symmetric or asymmetric dynamic changes.

ACKNOWLEDGMENT

The authors would like to thank Dr. R. Tínos for providing the source code of the EIGA-GAPX algorithm.

REFERENCES

[1] D. Angus and T. Hendtlass, "Ant colony optimisation applied to a dynamically changing problem," in *Developments in Applied Artificial Intelligence* (LNCS 2358). Heidelberg, Germany: Springer, 2002, pp. 618–627.

[2] D. Apiletti, E. Baralis, and T. Cerquitelli, "Energy-saving models for wireless sensor networks," *Knowl. Inf. Syst.*, vol. 28, no. 3, pp. 615–644, 2011.

[3] J. J. Bentley, "Fast algorithms for geometric traveling salesman problems," *ORSA J. Comput.*, vol. 4, no. 4, pp. 387–411, 1992.

[4] U. Boryczka and Ł. Strąk, "A hybrid discrete particle swarm optimization with pheromone for dynamic traveling salesman problem," in *Computational Collective Intelligence, Technologies and Applications* (LNCS 7654). Heidelberg, Germany: Springer, 2012, pp. 503–512.

[5] U. Boryczka and Ł. Strąk, "Efficient DPSO neighbourhood for dynamic traveling salesman problem," in *Computational Collective Intelligence, Technologies and Applications* (LNCS 8083). Heidelberg, Germany: Springer, 2013, pp. 721–730.

[6] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Proc. IEEE Congr. Evol. Comput.*, vol. 3. Washington, DC, USA, 1999, pp. 1875–1882.

[7] J. Cirasella, D. S. Johnson, L. A. McGeoch, and W. Zhang, "The asymmetric traveling salesman problem: Algorithms, instance generators, and tests," in *Algorithm Engineering and Experimentation* (LNCS 2153). Heidelberg, Germany: Springer, 2001, pp. 32–59.

[8] W. J. Cook, *In Pursuit of the Traveling Salesman: Mathematics at the Limit of Computation*. Princeton, NJ, USA: Princeton Univ. Press, 2011.

[9] G. A. Croes, "A method for solving traveling-salesman problems," *Oper. Res.*, vol. 6, no. 6, pp. 791–812, 1958.

[10] C. Cruz, J. R. Gonzalez, and D. A. Pelta, "Optimization in dynamic environments: A survey on problems, methods and measures," *Soft Comput.*, vol. 15, no. 7, pp. 1427–1448, 2011.

[11] W. Dazhi and L. Shixin, "An agent-based evolutionary search for dynamic travelling salesman problem," in *Proc. WASE Int. Conf. Inf. Eng. (ICIE)*, vol. 1. 2010, pp. 111–114.

[12] M. Dorigo, "Ant colony optimization," *Scholarpedia*, vol. 2, no. 3, 2007, Art. no. 1461.

[13] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, Apr. 1997.

[14] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996.

[15] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA, USA: MIT Press, 2004.

[16] C. J. Eyckelhof and M. Snoek, "Ant systems for a dynamic TSP," in *Ant Algorithms* (LNCS 2463). Heidelberg, Germany: Springer, 2002, pp. 88–99.

[17] P. M. França, M. Gendreau, G. Laporte, and F. M. Müller, "The *m*-traveling salesman problem with minmax objective," *Transp. Sci.*, vol. 29, no. 3, pp. 267–275, 1995.

[18] P. M. França, M. Gendreau, G. Laporte, and F. M. Müller, "A Tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times," *Int. J. Prod. Econ.*, vol. 43, nos. 2–3, pp. 79–89, 1996.

[19] L. M. Gambardella and M. Dorigo, "Ant-Q: A reinforcement learning approach to the traveling salesman problem," in *Proc. Int. Conf. Mach. Learn.*, Burlington, MA, USA, 1995, pp. 252–260.

[20] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. San Francisco, CA, USA: Freeman, 1979.

[21] M. Gendreau, A. Hertz, and G. Laporte, "New insertion and postoptimization procedures for the traveling salesman problem," *Oper. Res.*, vol. 40, no. 6, pp. 1086–1094, 1992.

[22] L. B. Gueta, R. Chiba, J. Ota, T. Arai, and T. Ueyama, "A practical and integrated method to optimize a manipulator-based inspection system," in *Proc. IEEE Int. Conf. Robot. Biomim.*, Sanya, China, 2007, pp. 1911–1918.

[23] M. Guntsch and M. Middendorf, "Pheromone modification strategies for ant algorithms applied to dynamic TSP," in *Proc. EvoWorkshops Appl. Evol. Comput.* (LNCS 2037). Como, Italy, 2001, pp. 213–222.

[24] M. Guntsch and M. Middendorf, "Applying population based ACO to dynamic optimization problems," in *Ant Algorithms* (LNCS 2463). Heidelberg, Germany: Springer, 2002, pp. 111–122.

[25] M. Guntsch, M. Middendorf, and H. Schmeck, "An ant colony optimization approach to dynamic TSP," in *Proc. Genet. Evol. Comput. Conf.*, San Francisco, CA, USA, 2001, pp. 860–867.

[26] Z.-C. Huang, X.-L. Hu, and S.-D. Chen, "Dynamic traveling salesman problem based on evolutionary computation," in *Proc. IEEE Congr. Evol. Comput.*, vol. 2. Seoul, South Korea, 2001, pp. 1283–1288.

[27] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments—A survey," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 303–317, Jun. 2005.

[28] D. S. Johnson *et al.*, "Experimental analysis of heuristics for the ATSP," in *The Traveling Salesman Problem and Its Variations* (Combinatorial Optimization), vol. 12, G. Gutin and A. P. Punnen, Eds. New York, NY, USA: Springer, 2007, pp. 445–487.

[29] R. Jonker and T. Volgenant, "Transforming asymmetric into symmetric traveling salesman problems," *Oper. Res. Lett.*, vol. 2, no. 4, pp. 161–163, 1983.

[30] P. C. Kanellakis and C. H. Papadimitriou, "Local search for the asymmetric traveling salesman problem," *Oper. Res.*, vol. 28, no. 5, pp. 1086–1099, 1980.

[31] L. Kang, A. Zhou, B. McKay, Y. Li, and Z. Kang, "Benchmarking algorithms for dynamic travelling salesman problems," in *Proc. IEEE Congr. Evol. Comput.*, vol. 2. Portland, OR, USA, 2004, pp. 1286–1292.

[32] C. Li, M. Yang, and L. Kang, "A new approach to solving dynamic traveling salesman problems," in *Proc. 6th Int. Conf. Simulat. Evol. Learn.*, Hefei, China, 2006, pp. 236–243.
[33] J. Li, M. Zhou, Q. Sun, X. Dai, and X. Yu, "Colored traveling salesman problem," *IEEE Trans. Cybern.*, vol. 45, no. 11, pp. 2390–2401, Nov. 2015.
[34] W. Li, "A parallel multi-start search algorithm for dynamic traveling salesman problem," in *Experimental Algorithms* (LNCS 6630). Heidelberg, Germany: Springer, 2011, pp. 65–75.
[35] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Oper. Res.*, vol. 21, no. 2, pp. 498–516, 1973.
[36] S. Lin, "Computer solutions of the traveling salesman problem," *Bell Syst. Tech. J.*, vol. 44, no. 10, pp. 2245–2269, Dec. 1965.
[37] L. Liu, D. Wang, and S. Yang, "An immune system based genetic algorithm using permutation-based dualism for dynamic traveling salesman problems," in *Applications of Evolutionary Computing* (LNCS 5484). Heidelberg, Germany: Springer, 2009, pp. 725–734.
[38] M. Mavrovouniotis, F. M. Müller, and S. Yang, "An ant colony optimization based memetic algorithm for the dynamic travelling salesman problem," in *Proc. Genet. Evol. Comput. Conf.*, Madrid, Spain, 2015, pp. 49–56.
[39] M. Mavrovouniotis and S. Yang, "A memetic ant colony optimization algorithm for the dynamic travelling salesman problem," *Soft Comput.*, vol. 15, no. 7, pp. 1405–1425, 2011.
[40] M. Mavrovouniotis and S. Yang, "Adapting the pheromone evaporation rate in dynamic routing problems," in *Applications of Evolutionary Computation* (LNCS 7835). Heidelberg, Germany: Springer, 2013, pp. 606–615.
[41] M. Mavrovouniotis and S. Yang, "Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors," *Appl. Soft Comput.*, vol. 13, no. 10, pp. 4023–4037, 2013.
[42] M. Mavrovouniotis, S. Yang, and X. Yao, "A benchmark generator for dynamic permutation-encoded problems," in *Proc. 12th Int. Conf. Parallel Prob. Solving Nat. (PPSN XII)* (LNCS 7492). Taormina, Italy, 2012, pp. 508–517.
[43] M. Mavrovouniotis, S. Yang, and X. Yao, "Multi-colony ant algorithms for the dynamic travelling salesman problem," in *Proc. IEEE Symp. Comput. Intell. Dyn. Uncertain Environ. (CIDUE)*, Orlando, FL, USA, 2014, pp. 9–16.
[44] L. Melo, F. Pereira, and E. Costa, "Multi-caste ant colony algorithm for the dynamic traveling salesperson problem," in *Adaptive and Natural Computing Algorithms* (LNCS 7824). Heidelberg, Germany: Springer, 2013, pp. 179–188.
[45] M. Niendorf, P. T. Kabamba, and A. R. Girard, "Stability of solutions to classes of traveling salesman problems," *IEEE Trans. Cybern.*, vol. 46, no. 4, pp. 973–985, Apr. 2016.
[46] J. J. Pantrigo, A. Duarte, Á. Sánchez, and R. Cabido, "Scatter search particle filter to solve the dynamic travelling salesman problem," in *Evolutionary Computation in Combinatorial Optimization* (LNCS 3448). Heidelberg, Germany: Springer, 2005, pp. 177–189.
[47] H. N. Psaraftis, *Dynamic Vehicle Routing Problems*. New York, NY, USA: Elsevier, 1988, pp. 223–248.
[48] A. E. Rizzoli, R. Montemanni, E. Lucibello, and L. M. Gambardella, "Ant colony optimization for real-world vehicle routing problems—From theory to applications," *Swarm Intell.*, vol. 1, no. 2, pp. 135–151, 2007.
[49] R. Shang, K. Dai, L. Jiao, and R. Stolkin, "Improved memetic algorithm based on route distance grouping for multiobjective large scale capacitated arc routing problems," *IEEE Trans. Cybern.*, vol. 46, no. 4, pp. 1000–1013, Apr. 2016.
[50] C. Silva and T. A. Runkler, "Ant colony optimization for dynamic traveling salesman problems," in *Proc. ARCS Workshops*, 2004, pp. 259–266.
[51] A. Simões and E. Costa, "CHC-based algorithms for the dynamic traveling salesman problem," in *Proc. EvoAppl. Appl. Evol. Comput*. (LNCS 6624). Turin, Italy, 2011, pp. 354–363.
[52] D. Soler, E. Martínez, and J. C. Micó, "A transformation for the mixed general routing problem with turn penalties," *J. Oper. Res. Soc.*, vol. 59, no. 4, pp. 540–547, 2008.
[53] T. Stützle and H. Hoos, "$\mathcal{MAX}$-$\mathcal{MIN}$ ant system and local search for the traveling salesman problem," in *Proc. IEEE Int. Conf. Evol. Comput.*, Indianapolis, IN, USA, 1997, pp. 309–314.
[54] T. Stützle and H. H. Hoos, "$\mathcal{MAX}$-$\mathcal{MIN}$ ant system," *Future Gener. Comput. Syst.*, vol. 16, no. 9, pp. 889–914, 2000.
[55] G. Tao and Z. Michalewicz, "Inver-over operator for the TSP," in *Proc. 5th Int. Conf. Parallel Prob. Solving Nat. (PPSN V)* (LNCS 1498). Amsterdam, The Netherlands, 1998, pp. 803–812.
[56] R. Tinós, D. Whitley, and A. Howe, "Use of explicit memory in the dynamic traveling salesman problem," in *Proc. Annu. Conf. Genet. Evol. Comput.*, Vancouver, BC, Canada, 2014, pp. 999–1006.
[57] N. L. J. Ulder, E. H. L. Aarts, H.-J. Bandelt, P. J. M. van Laarhoven, and E. Pesch, "Genetic local search algorithms for the traveling salesman problem," in *Proc. 1st Int. Conf. Parallel Prob. Solving Nat*. (LNCS 496). Dortmund, Germany, 1991, pp. 109–116.
[58] L.-N. Xing, P. Rohlfshagen, Y.-W. Chen, and X. Yao, "A hybrid ant colony optimization algorithm for the extended capacitated arc routing problem," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 41, no. 4, pp. 1110–1123, Aug. 2011.
[59] S. Yang, Y. Jiang, and T. T. Nguyen, "Metaheuristics for dynamic combinatorial optimization problems," *IMA J. Manag. Math.*, vol. 24, no. 4, pp. 451–480, 2013.
[60] A. Younes, P. Calamai, and O. Basir, "Generalized benchmark generation for dynamic combinatorial problems," in *Proc. Genet. Evol. Comput. Conf.*, Washington, DC, USA, 2005, pp. 25–31.
[61] A. Zhou, L. Kang, and Z. Yan, "Solving dynamic TSP with evolutionary approach in real time," in *Proc. IEEE Congr. Evol. Comput.*, vol. 2. Canberra, ACT, Australia, 2003, pp. 951–957.

Michalis Mavrovouniotis (M'13) received the B.Sc. degree in computer science from the University of Leicester, Leicester, U.K., in 2008, the M.Sc. degree in natural computation from the University of Birmingham, Birmingham, U.K., in 2009, and the Ph.D. degree in computer science from the University of Leicester, in 2013.

He is currently a Research Associate with the Centre for Computational Intelligence, De Montfort University, Leicester. His current research interests include evolutionary computation, swarm intelligence, memetic computing, combinatorial optimization problems, computational intelligence in dynamic and uncertain environments, and relevant real-world applications.

Felipe M. Müller received the B.Sc. degree in electrical engineering from the Federal University of Santa Maria, Santa Maria, Brazil, in 1987, and the M.Sc. and Ph.D. degrees in electrical engineering (automation) UNICAMP, Campinas, Brazil, in 1990 and 1993, respectively.

From 1991 to 1992, he was a Visiting Researcher with the Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, Montreal, QC, Canada. From 2014 to 2015, he was a Visiting Professor with the Centre for Computational Intelligence, De Montfort University, Leicester, U.K. He is currently a Titular Professor with the Applied Computing Department, Federal University of Santa Maria, where he was a Rector from 2009 to 2013. He has co-authored over 30 publications. His current research interests include heuristics, matheuristics, evolutionary algorithms, dynamic optimization problems and their applications.

Dr. Müller is a member of the Brazilian Society of Operations Research.

Shengxiang Yang (M'00–SM'14) received the B.Sc. and M.Sc. degrees in automatic control and the Ph.D. degree in systems engineering from Northeastern University, Shenyang, China, in 1993, 1996, and 1999, respectively.

He is currently a Professor in Computational Intelligence and the Director of the Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University, Leicester, U.K. He has over 210 publications. His current research interests include evolutionary and genetic algorithms, swarm intelligence, computational intelligence in dynamic and uncertain environments, artificial neural networks for scheduling, and relevant real-world applications.

Prof. Yang is the Chair of the Task Force on Evolutionary Computation in Dynamic and Uncertain Environments, under the Evolutionary Computation Technical Committee of the IEEE Computational Intelligence Society and the Founding Chair of the Task Force on Intelligent Network Systems, under the Intelligent Systems Applications Technical Committee of the IEEE Computational Intelligence Society.