Effective ACO-Based Memetic Algorithms for Symmetric and Asymmetric Dynamic Changes

Michalis Mavrovouniotis*, Iaê S. Bonilha†, Felipe M. Müller‡, Georgios Ellinas*, Marios Polycarpou*
*KIOS Research and Innovation Center of Excellence,
Department of Electrical and Computer Engineering, University of Cyprus, 2109 Nicosia, Cyprus.
email: {mavrovouniotis.michalis,gellinas,mpolycar}@ucy.ac.cy
†PPGEP,
Federal University of Santa Maria, 97105-900, Santa Maria, Brazil.
email: iaesb@hotmail.com
‡Department of Applied Computing,
Federal University of Santa Maria, 97105-900, Santa Maria, Brazil.
email: felipe@inf.ufsm.br

***Abstract*—Ant colony optimization (ACO) algorithms have proved to be suitable for solving dynamic optimization problems (DOPs). The integration of local search operators with ACO has also proved to significantly improve the output of ACO algorithms. However, almost all previous works of ACO in DOPs do not utilize local search operators. In this work, the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS), one of the best ACO variations, is integrated with advanced and effective local search operators, i.e., the Lin-Kernighan and the Unstringing and Stringing heuristics, resulting in powerful memetic algorithms. The best solution constructed by ACO is passed to the operator for local search improvements. The proposed memetic algorithms aim to combine the adaptation capabilities of ACO for DOPs and the superior performance of the local search operators. The travelling salesperson problem is used as the base problem to generate both symmetric and asymmetric dynamic test cases. Experimental results show that the $\mathcal{MM}$AS is able to provide good initial solutions to the local search operators especially in the asymmetric dynamic test cases.**

***Index Terms*—Ant colony optimization, local search, memetic algorithm, dynamic travelling salesperson problem**

I. Introduction

Ant colony optimization (ACO) algorithms have proved that they are powerful problem solving tools. They are able to provide the optimal (or near to the optimal) solution for difficult combinatorial optimization problems (e.g., the travelling salesperson problem (TSP) [1]). Traditionally, researchers have focused their attention on static optimization problems, where the environment of the problem remains fixed during the optimization process of an algorithm. However, many real-world applications are subject to dynamic environments. Dynamic optimization problems (DOPs) are challenging since the aim of an algorithm is not only to locate the optimum of the problem quickly, but to efficiently track the moving optimum when changes occur [2]. A dynamic change may involve factors like the objective function, input variables, problem instance, and constraints.

The integration of local search operators with ACO (or other metaheuristics), resulting to the so-called *memetic algorithms*, showed that it significantly improves the output in static [5], [6] and, recently, in DOPs [14]. This is because ACO-based memetic algorithms can better explore locally a neighbourhood in the search space compared to conventional ACO algorithms. In addition, ACO-based memetic algorithms take advantage of the global search capabilities of ACO to guide local search in neighbourhoods, leading to high quality solutions. Furthermore, ACO-based memetic algorithms in DOPs inherit the adaptation capabilities of ACO when dynamic changes occur. In this paper, we consider two advanced local search operators: 1) Lin-Kernighan (LK) [8], and 2) Unstringing and Stringing (US) [9]. Although the US operator has already been investigated in the ACO-based memetic algorithm in [7], [14], in this work the investigation is extended with comparisons against algorithms based on the advanced LK operator.

TSP is used as the base problem to systematically generate dynamic test cases [11]. Almost all of the dynamic test cases considered in the literature focused only on symmetric dynamic changes [7], [11], [18]. However, the real-world is often asymmetric [12], [14]. For example, in transportation routing problems the time when driving in one direction is not necessarily the same in the opposite direction (e.g., may be affected by different traffic conditions). Hence, in this work, we consider both symmetric and asymmetric dynamic changes. The rest of the paper is organized as follows. Section II describes the TSP and the construction of the dynamic test cases. Section III describes the memetic framework based on one of the best variations of ACO. The core logic of the two local search operators is also described to make the paper self-contained. Section IV gives the experimental results and analysis. Finally, Section V concludes this paper.

II. Dynamic Test Environments

A. Base Problem Formulation

TSP is used as the base problem to generate dynamic test cases. Typically, a TSP instance is modelled by a fully connected weighted graph $G = (N, A)$, where $N = \{1, \ldots, n\}$ is a set of n nodes and $A = \{(i, j) \mid i, j \in N, i \neq j\}$ is a set of arcs. For the classic TSP, nodes and arcs represent

the cities and the links between them. Each arc $(i, j) \in A$ is associated with a non-negative value $w_{ij} \in \mathbb{R}^+$, which for the classic TSP represents the distance between nodes i and j. For symmetric TSPs, these distances are independent of the direction of traversing the arcs, that is, $w_{ij} = w_{ji}$ for every pair of nodes. If $w_{ij} \neq w_{ji}$ for at least one pair of nodes, then the TSP becomes asymmetric. Every problem instance consists of a weight matrix, i.e., $\mathbf{W} = \{w_{ij}\}_{n\times n}$, that contains all the weights associated with the arcs of the corresponding graph G.

B. Generating Dynamic Test Environments

To generate dynamic test cases, the weight matrix of the problem instance becomes dynamic as follows [11]:

$$\mathbf{W}(T) = \{w_{ij}(T)\}_{n\times n}, \tag{1}$$

where $T = \lceil t/f \rceil$ is the period of a dynamic change, t is the evaluation count and f is the frequency of change. Note that the introduced dynamic changes are synchronized with the optimization process of the algorithm. Hence, the parameter f is expressed in algorithmic evaluations.

A particular dynamic test case can be generated by assigning an increasing or decreasing factor value to the arc connecting nodes i and j as follows:

$$w_{ij}(T+1) = \begin{cases} w_{ij}(0) + \mathcal{R}_{ij}, & \text{if arc } (i,j) \in A_S(T), \\ w_{ij}(T), & \text{otherwise,} \end{cases} \tag{2}$$

where $w_{ij}(0)$ is the initial weight of the arc connecting nodes i and j (i.e., from the static problem instance when $T = 0$), $\mathcal{R}_{ij}$ is a normally distributed random number (with zero mean and standard deviation set to $0.2 \cdot w_{ij}(0)$ [12]) that defines the modified factor value of the arc, $A_S(T) \subset A$ defines the set of arcs randomly selected for the change at that period and T defines the environmental period index as defined in Eq. (1).

The size of the set A is defined by the number of arcs (i.e., $n(n-1)$ for asymmetric cases and $n(n-1)/2$ for symmetric cases). Hence, the size of $A_S(T)$ is defined by the magnitude of change (i.e., $m \in [0, 1]$) and the size of A. For example, at period T, exactly $\lceil mn(n-1) \rceil$ and $\lceil mn(n-1)/2 \rceil$ arcs will be selected to change their weights in asymmetric and symmetric cases, respectively. The higher the value of m, the more arcs will be selected for changes. Note that the arcs for asymmetric cases are directed whereas for symmetric cases are undirected. Therefore, if w_{ij} changes in symmetric cases, then w_{ji} changes uniformly (i.e., $w_{ji} = w_{ij}$). On the contrary, when w_{ij} changes in asymmetric cases, w_{ji} will not change unless arc (j, i) is selected for a change (but not necessarily uniformly with w_{ij}).

A particular solution $\pi = [\pi_1, \ldots, \pi_n]$ in the search space is specified by a permutation of the node indices, and for the dynamic TSP (DTSP), it is evaluated as follows:

$$\phi(\pi, t) = w_{\pi_n \pi_1}(T) + \sum_{i=1}^{n-1} w_{\pi_i \pi_{i+1}}(T). \tag{3}$$

III. ACO-Based Memetic Algorithms

The ACO metaheuristic consists of a colony of ω ants that construct solutions and share their information among each other via their pheromone trails. One of the best performing ACO variations, i.e., the $\mathcal{MAX}$-$\mathcal{MIN}$ AS ($\mathcal{MM}$AS) [13], is used in the memetic framework shown in Algorithm 1. Since the TSP is used as the base problem to generate dynamic test cases, it is used as a concrete example to describe the ACO-based memetic framework.

A. Constructing Solutions

Ants read pheromones to construct solutions and write pheromones to mark their constructed solutions. The probability with which ant k, currently at node i, will move to node j is calculated as follows:

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

where τ_{ij} and η_{ij} are the existing pheromone trail and the heuristic information available a priori between nodes i and j, respectively. The pheromone trails are initialized uniformly with a value τ_0. The heuristic information is calculated as $\eta_{ij} = 1/w_{ij}(T)$ where $w_{ij}(T)$ is defined as in Eq. (1). $\mathcal{N}_i^k$ is the set of unvisited nodes for ant k adjacent to node i. α and β are the two parameters which determine the relative influence of τ_{ij} and η_{ij}, respectively.

B. Applying Local Search Operator

Stützle and Hoos [13] applied local search operators to the iteration-best ant of $\mathcal{MM}$AS after every iteration, whereas in [5], they further applied local search operators to all ants. Considering that local search operators are computationally expensive methods, such an extensive usage may not be very efficient for DTSP because the computation time naturally increases. As previously discussed, for DTSPs, algorithms must produce high quality solutions quickly [12]. In [14] the local search operator is applied to the the best-so-far ant of $\mathcal{MM}$AS only when a new best solution is found. This is because local search operators are typically 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 (and waste evaluations) to potentially "improve" a solution for which basically no further improvement is possible. In this work we investigate two powerful local search heuristics designed for the TSP, i.e., LK and US heuristics, described below.

1) Lin-Kernighan: The LK heuristic performs a series of λ-opt moves to transform a TSP tour into a shorter one [8]. A λ-opt move consists of the exchange of a set of λ tour arcs by a set of λ new arcs. The LK heuristic starts with two empty arc sets: X (i.e., *out*-arcs) and Y (i.e., *in*-arcs). At each step one arc that currently belongs to the tour will be added to X and a new arc that does not belong to the tour will be added to Y. After the first step, the LK heuristic will favour arc insertions that result in a shorter complete TSP tour. When

Algorithm 1 ACO-Based Memetic Framework

```
1: InitializePheromoneTrails(τ_0)
2: while (termination condition not satisfied) do
3:    ConstructSolutions
4:    π^ib ← FindIterationBest
5:    if (φ(π^ib, t) < φ(π^bs, t)) then
6:       π^bs ← π^ib
7:       ApplyLocalSearch(π^bs)
8:    end if
9:    PheromoneUpdate
10: end while
11: OUTPUT: π^bs    %best TSP solution
```

a new complete tour is achieved, the algorithm will begin a new phase of arc exchanges and this process will continue until there is no further improvement.

A set of rules, that each step must follow, was established in order to enhance the algorithm's efficiency as follows:

- Each arc removed must share a node with its added counterpart. After the first arc exchange in each cycle, each arc being removed must also share a node with the previously added arc. Fig. 1 illustrates an example, where on the first step arc (V_1,V_2) is removed and arc (V_2,V_4), which shares the node V_2 with its removed counterpart, is added. On the second step arc (V_3,V_4), which shares node V_4 with the previously added arc, is removed and arc (V_3,V_1) is added, closing the tour.
- No exchanges that result in the tour being broken into multiple closed circuits are allowed. An example of this type of exchange is shown in Fig. 2, where arcs (V_1,V_2) and (V_3,V_4) are removed and arcs (V_4,V_1) and (V_3,V_2) are added. In this case, the addition of either of the arcs would not be accepted because it will result in a segment of tour forming a cycle.
- Each pair of arcs exchanged must be gainful, meaning that each arc being added must be shorter than its removed counterpart.
- Once an arc is removed, until the tour is closed again, it cannot be added again in subsequent exchanges.

Although LK was originally designed for symmetric problems [8], it can be also applied to asymmetric problems by transforming an asymmetric weight matrix to a symmetric weight matrix (i.e., by doubling the nodes of the graph) [15], [22].

2) Unstringing and Stringing: The US heuristic is based on the removal (or unstringing) of nodes from the tour and their subsequent re-insertion (or stringing) [9]. The main feature of the algorithm is that the re-insertion of nodes can happen between non-adjacent nodes, resulting in a tour where both nodes become adjacent to the node being inserted. Suppose that we wish to insert V_x between any two nodes V_i and V_j. For a given orientation of a tour, consider V_k a node in the subtour from V_j to V_i, and V_l a node in the subtour from V_i to V_j. We also consider for any node V_h on the tour, V_{h+1}

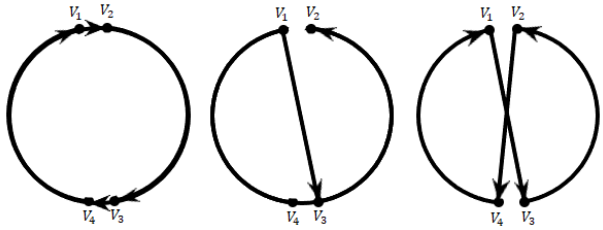

Fig. 1: A 2–*opt* move.

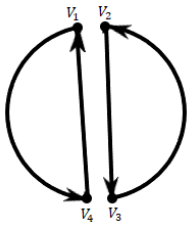

Fig. 2: Closed circuits.

its successor and V_{h-1} its predecessor. The re-insertion of V_x between V_i and V_j can be done in several ways using different types of insertions and removals. In [7], [9], only symmetric problem instances were considered and tackled with Type I and Type II removals (Fig. 3(a) and Fig. 3(b)) and Type I and Type II insertions (Fig. 4(a) and Fig. 4(b)). In [10], [14] another two types of removals and another two types of insertions were considered in order to tackle asymmetric problem instances: Type III and Type IV removals (Fig. 3(c) and Fig. 3(d)), and Type III and Type IV insertions (Fig. 4(c) and Fig. 4(d)).

The unstringing procedure removes a given node from the tour and repairs the connections with the remaining nodes in order to have a closed tour. The procedure consists of four types of removals as follows:

- Type I removal: Assume that V_j belongs to the neighbourhood of V_{i+1} and V_k belongs to the neighbourhood 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.
- Type II removal: Assume that V_j belongs to the neighbourhood of V_{i+1}, V_k belongs to the neighbourhood 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 neighbourhood 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.
- Type III removal: Assume that V_j belongs to the neighborhood of V_{i+1} and V_k belongs to the neighborhood of

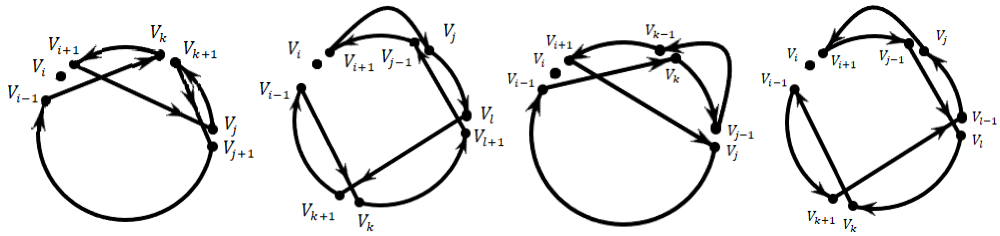

Fig. 3: (a) Type I removal, (b) Type II removal, (c) Type III removal, and (d) Type IV removal.

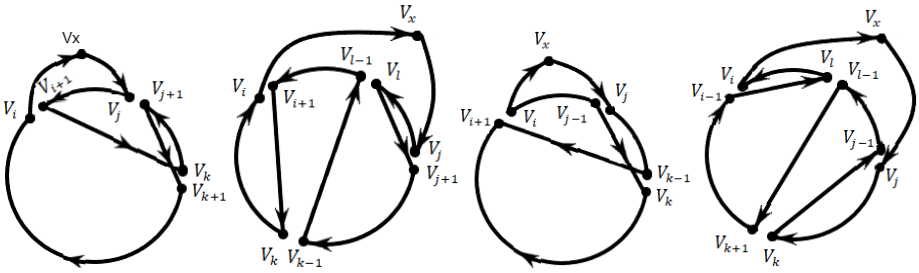

Fig. 4: (a) Type I insertion, (b) Type II insertion, (c) Type III insertion, and (d) Type IV insertion.

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.

- 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.

The stringing procedure is basically the reverse of the unstringing procedure and consists of four types of insertions as follows:

- 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.
- 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.
- 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 a way that almost the entire 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.
- Type IV insertion: Similarly, 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.

C. Updating Pheromones

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

$$\tau_{ij} \leftarrow (1 - \rho)\, \tau_{ij}, \forall (i,j) \in A, \tag{5}$$

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:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best}, \forall (i,j) \in \pi^{best}, \tag{6}$$

where $\Delta\tau_{ij}^{best} = 1/\phi(\pi^{best}, t)$ is the amount of pheromone that the best ant deposits. The best ant that is allowed to deposit pheromone may be either the best-so-far ant[1], in which case $\pi^{best} = \pi^{bs}$, or the iteration-best ant, in which case $\pi^{best} = \pi^{ib}$. Both ants are used to deposit pheromones to achieve a transition from a stronger exploration of the search space early to a stronger exploitation of the best-so-far solution later. More precisely, let f^{bs} indicate the frequency the best-so-far ant is allowed to deposit pheromone. f^{bs} increases as the search progresses following a pre-defined schedule [5]:

$$f^{bs} = \begin{cases} \infty, & \text{if } I \leq 25, \\ 5, & \text{if } 26 \leq I \leq 75, \\ 3, & \text{if } 76 \leq I \leq 125, \\ 2, & \text{if } 126 \leq I \leq 250, \\ 1, & \text{otherwise,} \end{cases} \tag{7}$$

where f^{bs} is the number of algorithmic iterations between two updates performed by the best-so-far ant and I is the iteration counter of the algorithm. In other words, the emphasis from the iteration-best ant to the best-so-far ant for the pheromone update is shifted gradually. The schedule is restarted at the beginning of every dynamic change. It must be noted that pheromones are updated after the local search improvements to mark them in the pheromone trails so they can be exploited in the following iterations.

D. Maintaining Diversity

Maintaining the diversity of the constructed solutions is one of the key factors when addressing DOPs. This is because it helps the search to escape from (outdated) solutions of previously optimized environments and adapt to the new ones [20].

Since only the best ant is allowed to deposit pheromone, the search may quickly converge towards the best solution found in the first iterations. Therefore, the pheromone trails are occasionally reinitialized to the current τ_{max} value to increase exploration. For example, whenever the stagnation behaviour[2] occurs or when no improved solution is found for a given number of iterations, the pheromone reinitialization mechanism is triggered.

In addition, the lower and upper limits τ_{min} and τ_{max} of the pheromone trail values are imposed. In this way, the probability of selecting an arc will always be $p_{ij}^k > 0$, and, consequently, all arcs will have a chance to be selected. The τ_{max} value is set to $\tau_{max} = 1/(\rho \cdot \phi(\pi^{bs}, t))$, and is updated whenever a new best-so-far ant is found. The τ_{min} value is set to $\tau_{min} = \tau_{max}/(2n)$, where n is the number of nodes.

[1]The best-so-far ant is a special ant that may not necessarily belong in the current constructing colony.

[2]Detected using λ-branching [16] that calculates the statistics regarding the distribution of the current pheromone trails.

E. Responding to Dynamic Changes

ACO algorithms are able to use knowledge from previous environments via their pheromone trails and can be applied directly to DOPs without any modifications [17], [18]. For example, when the changing environments are similar, the pheromone trails of the previous environment may provide knowledge to speed up the optimization process to the new environment. However, the algorithm must be flexible enough to accept the knowledge transferred from the pheromone trails, or eliminate the pheromone trails, in order to adapt well to the new environment. When a dynamic change occurs, evaporation eliminates the pheromone trails of the previous environment from areas that are generated on the old optimum and helps ants to explore areas for the new optimum.

In case the changing environments are completely different, then pheromone reinitialization may be a better choice rather than transferring the knowledge from previous pheromone trails [17]–[19].

IV. Experimental Studies

A. Experimental Setup

In the experiments, we investigate the effects of having ACO providing its solutions for symmetric and asymmetric dynamic changes rather than using randomly generated solutions for local search improvements. Comparisons of two effective local search operators are performed. In particular, the performance of the following algorithms is investigated:

- $\mathcal{MM}$AS+US: the US operator is applied in $\mathcal{MM}$AS whenever a new best-so-far ant is found until there is no further improvement.
- $\mathcal{MM}$AS+LK: the LK operator is applied in $\mathcal{MM}$AS whenever a new best-so-far ant is found until there is no further improvement.
- US: the US operator applied on a random initial solution rather than the best-so-far solution generated by $\mathcal{MM}$AS.
- LK: the LK operator applied on a random initial solution rather than the best-so-far solution generated by $\mathcal{MM}$AS.

All algorithmic parameters were set to commonly used values: $\alpha = 1$, $\beta = 5$, $\rho = 0.8$ and the number of ants was set to $\omega = 50$. DTSPs are generated from five static benchmark instances obtained from TSPLIB[3]: pcb442, u574, pcb1173, rat783, lin318 using the dynamic generator described in Section II. The first three benchmark instances arise from the task of drilling holes in printed circuit boards, the next benchmark instance arise from rattled grid, and the last benchmark instance from the travel cost between cities. The frequency of change f was set to change every $10e4$ algorithmic evaluations and the magnitude of change m was set to 0.05, 0.1, 0.2 and 0.4, indicating small to medium changing environments. Totally, a series of 4 DTSP test cases were constructed from each stationary instance, for

[3]Available from http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

symmetric and asymmetric changes, to systematically analyze the performance of the algorithms (all asymmetric problem instances have an extension of `.atsp` at the end of the problem label). For each algorithm on a DTSP, 30 independent runs were executed on the same set of random seed numbers. For each run, 100 environmental changes were allowed and an observation (i.e., the value of the best-so-far ant after a dynamic change) was recorded. For a fair comparison, all the algorithms performed the same number of evaluations. The proportional evaluations required when applying US and LK operators are added to the total evaluations of the algorithms.

The *offline performance* [21] was used to evaluate the overall performance of the algorithms, which is defined as:

$$\bar{P}_{offline} = \frac{1}{E}\sum_{t=1}^{E}\phi(\pi^{bs}, t), \tag{8}$$

where E is the total number of evaluations and π^{bs} is the best-so-far solution quality after a change.

B. Experimental Results and Discussion

The experimental results regarding the offline performance of the investigated algorithms for all DTSPs are presented in Table I. The corresponding statistical results are presented in Table II, in which pairwise *Mann–Whitney* statistical tests with a significance level of 0.05 were performed. In Table II, the results are shown as "+", "−" and "∼" when the first algorithm is significantly better than the second one, when the second algorithm is significantly better than the first one, and when the two algorithms are not significantly different, respectively. In Fig. 5 and Fig. 6, the dynamic average offline performance against the algorithmic iterations of $\mathcal{MM}$AS+US, $\mathcal{MM}$AS+LK, US and LK are plotted for the last ten environmental changes to better understand the behaviour of the algorithms in symmetric and asymmetric dynamic changes, respectively. From the experimental results the following observations can be drawn.

First, $\mathcal{MM}$AS+US significantly outperforms US in both symmetric and asymmetric cases (see the comparisons in Table II). This is because $\mathcal{MM}$AS can provide the US local search heuristic an initial solution from a promising neighbourhood (probably the one that contains the global optimum solution) in the search space, whereas it is less likely when starting from an initial random solution as in the US algorithm. On the contrary, $\mathcal{MM}$AS+LK significantly outperforms LK only in asymmetric DTSPs, but it has no effect in symmetric DTSPs (since the improvement is not significant); see the comparisons in Table II. It is well known that the LK local search heuristic is considered by far the best heuristic on symmetric cases [22], and, consequently, starting from a random initial solution (as in the LK) will still result in good performance. However, it is still interesting to observe that the guidance provided from $\mathcal{MM}$AS is very effective for the LK heuristic in asymmetric DTSPs. This is because the LK heuristic was designed specifically for symmetric cases, and, thus, loses its effectiveness in asymmetric cases.

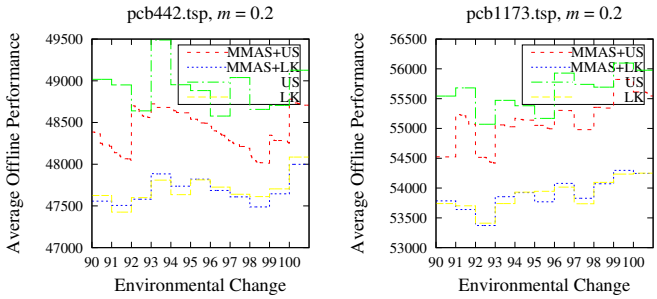

Fig. 5: Dynamic average offline performance of algorithms for symmetric DTSPs with $m = 0.2$.

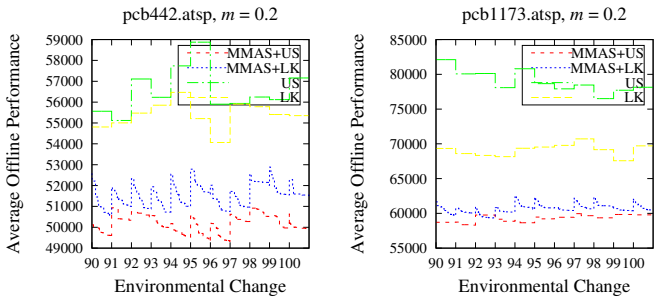

Fig. 6: Dynamic average offline performance of algorithms for asymmetric DTSPs with $m = 0.2$.

However, with the guidance of $\mathcal{MM}$AS it will still explore some promising neighbourhoods.

Second, $\mathcal{MM}$AS+US significantly outperforms $\mathcal{MM}$AS+LK in most asymmetric cases whereas $\mathcal{MM}$AS+LK significantly outperforms $\mathcal{MM}$AS+US in most symmetric cases. This is because $\mathcal{MM}$AS+US performs a wide range of alternative untried moves that tend to improve the solution quality of the tour. However, when the LK heuristic is dealing with asymmetric cases it performs a very specific set of moves that preserve the direction of the tour. The moves are composed by a specific case of 3–*opt* (i.e., cycle patching) and non-sequential 4–*opt* (i.e., the so-called double-bridge move) moves. On the contrary, when the US heuristic is dealing with asymmetric cases the segments of the tour in which their direction is not preserved will be recalculated. In this way, the set of moves performed by the US heuristic will not be restricted only to the moves that preserve the direction of the tour. In addition, Type III and Type IV moves are designed to cope with asymmetric cases, by performing moves in the opposite direction of Type I and Type II moves, hoping that new regions (in the search) with high quality solutions will be found.

V. Conclusions

In this paper, we integrate two advanced local search operators (i.e., LK and US) with the $\mathcal{MM}$AS for dynamic environments. The aim of the integration is to take advantage of the adaptation capabilities of $\mathcal{MM}$AS and the solution

TABLE I: Experimental results regarding the average offline performance of algorithms on symmetric DTSPs (upper half) and asymmetric DTSPs (lower half).

Problem Instance	m	$\mathcal{MMAS}$+US	US	$\mathcal{MMAS}$+LK	LK
lin318.tsp	0.05	39802.1	39934.9	38541.1	38557.7
	0.1	39677.8	39772.2	38435.6	38453.8
	0.2	39529.4	39533.8	38174.9	38175.5
	0.4	39718.2	39666.2	38271.6	38286.7
pcb442.tsp	0.05	48944.0	49437.6	48228.4	48267.2
	0.1	48736.3	49236.6	48021.4	48037.1
	0.2	48784.5	49225.7	48012.2	48041.8
	0.4	48653.7	49022.0	47872.9	47893.2
u574.tsp	0.05	35882.8	35798.7	34553.6	34564.8
	0.1	35429.7	35380.0	34203.7	34211.0
	0.2	35332.2	35254.3	34055.4	34065.3
	0.4	35173.6	35134.4	33900.1	33911.2
rat783.tsp	0.05	8194.1	8277.4	8024.4	8028.7
	0.1	8124.9	8204.9	7955.6	7956.6
	0.2	8071.6	8148.8	7900.9	7900.2
	0.4	8094.5	8165.1	7914.4	7914.2
pcb1173.tsp	0.05	55827.9	56390.2	54532.4	54569.0
	0.1	55520.2	56022.3	54182.9	54203.7
	0.2	55118.9	55637.8	53844.5	53833.5
	0.4	55275.9	55770.4	53934.4	53950.8
lin318.atsp	0.05	40526.1	45341.6	40930.7	44547.6
	0.1	40904.1	45825.3	41256.4	44204.1
	0.2	40836.0	45761.4	41123.0	43958.1
	0.4	41024.6	45861.7	41208.3	43662.3
pcb442.atsp	0.05	50082.9	55346.8	51402.0	56071.5
	0.1	49973.3	55876.8	51170.4	55366.6
	0.2	50196.9	56118.0	51429.9	55352.0
	0.4	50360.8	56349.9	51307.5	55136.3
u574.atsp	0.05	37446.5	42870.1	37795.9	40794.4
	0.1	37527.1	43737.7	37656.7	40299.6
	0.2	37644.8	44139.8	37553.0	40412.2
	0.4	37703.5	44098.4	37465.9	40106.1
rat783.atsp	0.05	8628.0	10210.5	8880.7	9665.1
	0.1	8575.9	10360.7	8754.0	9497.0
	0.2	8575.6	10381.2	8727.2	9455.5
	0.4	8589.5	10413.6	8691.7	9420.3
pcb1173.atsp	0.05	59529.5	76943.1	61201.9	69814.8
	0.1	59466.5	78057.9	61075.6	69118.8
	0.2	59240.7	79078.1	60776.9	68675.6
	0.4	59543.0	80021.3	60836.5	68712.6

improvement of the local search operators. The travelling salesperson problem is used as the base problem to generate dynamic test cases, both with symmetric and asymmetric dynamic changes. The performance of the resulting memetic algorithms is investigated on dynamic test cases that are systematically constructed.

From the experimental results, the following conclusions can be drawn. First, $\mathcal{MM}$AS provides good initial points in the search space for both local search operators. Second, the integration of the local search operators with $\mathcal{MM}$AS is more effective on asymmetric DTSPs. Third, the US operator is more effective on asymmetric DTSPs, whereas the LK operator is more effective on symmetric DTSPs.

ACKNOWLEDGMENT

This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 739551 (KIOS CoE) and from the Government of the Republic of Cyprus through the Directorate General for European Programmes, Coordination and Development.

REFERENCES

[1] M. Dorigo and T. Stützle, *Ant colony optimization*. Cambridge, MA: MIT Press, 2004.

[2] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments–a survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[3] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.

[4] M. Mavrovouniotis and S. Yang, "A memetic ant colony optimization algorithm for the dynamic travelling salesman problem," *Soft Computing*, vol. 15, no. 7, pp. 1405–1425, 2011.

[5] T. Stützle and H. H. Hoos, "$\mathcal{MAX}$–$\mathcal{MIN}$ ant system," *Future Generation Computer Systems*, vol. 16, no. 8, pp. 889–914, 2000.

[6] N. Ulder, E. Aarts, H.-J. Bandelt, P. van Laarhoven, and E. Pesch, "Genetic local search algorithms for the traveling salesman problem," in *Parallel Problem Solving from Nature*, ser. Lecture Notes in Computer Science, H.-P. Schwefel and R. Männer, Eds. Springer Berlin Heidelberg, 1991, vol. 496, pp. 109–116.

[7] M. Mavrovouniotis, F. M. Müller, and S. Yang, "An ant colony optimization based memetic algorithm for the dynamic travelling salesman problem," in *Proc. 2015 Genetic and Evol. Comput. Conf.*, 2015, pp. 49–56.

[8] S. Lin and B. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

TABLE II: Statistical results regarding the average offline performance of algorithms on symmetric DTSPs (upper half) and asymmetric DTSPs (lower half).

Problem Instance	m	$\mathcal{MM}$AS+US **vs** $\mathcal{MM}$AS+LK	$\mathcal{MMAS}$+US **vs** US	$\mathcal{MMAS}$+LK **vs** LK	US **vs** LK
lin318.tsp	0.05	−	+	∼	−
	0.1	−	+	∼	−
	0.2	−	∼	∼	−
	0.4	−	∼	∼	−
pcb442.tsp	0.05	−	+	∼	−
	0.1	−	+	∼	−
	0.2	−	+	∼	−
	0.4	−	+	∼	−
u574.tsp	0.05	−	∼	∼	−
	0.1	−	∼	∼	−
	0.2	−	∼	∼	−
	0.4	−	∼	∼	−
rat783.tsp	0.05	−	+	∼	−
	0.1	−	+	∼	−
	0.2	−	+	∼	−
	0.4	−	+	∼	−
pcb1173.tsp	0.05	−	+	∼	−
	0.1	−	+	∼	−
	0.2	−	+	∼	−
	0.4	−	+	∼	−
lin318.atsp	0.05	+	+	+	−
	0.1	+	+	+	−
	0.2	+	+	+	−
	0.4	+	+	+	−
pcb442.atsp	0.05	+	+	+	+
	0.1	+	+	+	−
	0.2	+	+	+	−
	0.4	+	+	+	−
u574.atsp	0.05	+	+	+	−
	0.1	∼	+	+	−
	0.2	∼	+	+	−
	0.4	−	+	+	−
rat783.atsp	0.05	+	+	+	−
	0.1	+	+	+	−
	0.2	+	+	+	−
	0.4	+	+	+	−
pcb1173.atsp	0.05	+	+	+	−
	0.1	+	+	+	−
	0.2	+	+	+	−
	0.4	+	+	+	−

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

[10] 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," *International Journal of Production Economics.*, vol. 43, no. 2–3, pp. 79–89, 1996.

[11] M. Mavrovouniotis and S. Yang, "Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors," *Applied Soft Computing*, vol. 13, no. 10, pp. 4023–4037, 2013.

[12] R. Tinós, D. Whitley, and A. Howe, "Use of explicit memory in the dynamic traveling salesman problem," in *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2014, pp. 999–1006.

[13] T. Stützle and H. Hoos, "$\mathcal{MAX}$–$\mathcal{MIN}$ ant system and local search for the traveling salesman problem," in *IEEE International Conference on Evolutionary Computation*, 1997, pp. 309–314.

[14] M. Mavrovouniotis, F. M. Müller, and S. Yang, "Ant colony optimization with local search for the dynamic travelling salesman problems," *IEEE Transactions on Cybernetics*, vol. 47, no. 7, pp. 1743–1756, 2017.

[15] R. Jonker and T. Volgenant, "Transforming asymmetric into symmetric traveling salesman problems," *Operations Research Letters*, vol. 2, no. 4, pp. 161–163, 1983.

[16] L. M. Gambardella and M. Dorigo, "Ant-Q: A reinforcement learning approach to the traveling salesman problem," in *International Conference on Machine Learning*, 1995, pp. 252–260.

[17] D. Angus and T. Hendtlass, "Ant colony optimisation applied to a dynamically changing problem," in *Developments in Applied Artificial Intelligence*, ser. Lecture Notes in Computer Science, T. Hendtlass and M. Ali, Eds. Springer Berlin Heidelberg, 2002, vol. 2358, pp. 618–627.

[18] M. Mavrovouniotis and S. Yang, "Adapting the pheromone evaporation rate in dynamic routing problems," in *Applications of Evolutionary Computation*, ser. Lecture Notes in Computer Science, A. Esparcia-Alcázar, Ed. Springer Berlin Heidelberg, 2013, vol. 7835, pp. 606–615.

[19] M. Guntsch and M. Middendorf, "Applying population based ACO to dynamic optimization problems," in *Ant Algorithms*, ser. Lecture Notes in Computer Science, M. Dorigo, G. Di Caro, and M. Sampels, Eds. Springer Berlin Heidelberg, 2002, vol. 2463, pp. 111–122.

[20] M. Mavrovouniotis, C. Li, and S. Yang, "A survey of swarm intelligence for dynamic optimization: Algorithms and applications," *Swarm and Evolutionary Computation*, vol. 33, pp. 1–17, 2017.

[21] J. Branke and H. Schmeck, "Designing evolutionary algorithms for dynamic optimization problems," in *Advances in Evolutionary Computing*, ser. Natural Computing Series, A. Ghosh and S. Tsutsui, Eds. Springer Berlin Heidelberg, 2003, pp. 239–262.

[22] K. Helsgaun, "An effective implementation of the LinKernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.