Ant Colony Optimization Algorithms for Dynamic Optimization: A Case Study of the Dynamic Travelling Salesperson Problem

Michalis Mavrovouniotis
KIOS Research and Innovation Center of Excellence, Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, CYPRUS

Shengxiang Yang
School of Computer Science and Informatics, De Montfort University, Leicester, UK

Mien Van
School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Belfast, UK

Changhe Li
School of Automation and Hubei Key Laboratory of Advanced Control and Intelligent Automation for Complex Systems, China University of Geosciences, Wuhan, CHINA

Marios Polycarpou
KIOS Research and Innovation Center of Excellence, Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, CYPRUS

Abstract

Ant colony optimization is a swarm intelligence metaheuristic inspired by the foraging behavior of some ant species. Ant colony optimization has been successfully applied to challenging optimization problems. This article investigates existing ant colony optimization algorithms specifically designed for combinatorial optimization problems with a dynamic environment. The investigated algorithms are classified into two frameworks: evaporation-based and population-based. A case study of using these algorithms to solve the dynamic travelling salesperson problem is described. Experiments are systematically conducted using a proposed dynamic benchmark framework to analyze the effect of important ant colony optimization features on numerous test cases. Different performance measures are used to evaluate the adaptation capabilities of the investigated algorithms, indicating which features are the most important when designing ant colony optimization algorithms in dynamic environments.

Corresponding Author: Michalis Mavrovouniotis (Email: mavrovouniotis.michalis@ucy.ac.cy)

I. Introduction

Ant colony optimization (ACO) is a well-known metaheuristic for combinatorial optimization problems inspired by the foraging behavior of real ant colonies [1], [2]. Ants are able to find the shortest path between a food source and their nest [3]. They lay pheromones (a chemical substance produced by ants) on the ground to mark their path, forming in this way a pheromone trail, and they tend to follow paths marked by strong pheromone concentrations. In this way, the ants are able to share information with their nest mates.

In ACO, a colony of (artificial) ants cooperates in constructing high-quality solutions to difficult combinatorial optimization problems [4]. The construction of solutions is guided by (artificial) pheromone trails. The pheromone model used to update the pheromone trails is inspired by the pheromone trail laying and following behavior of real ants. This is basically a parametrized probabilistic model that is modified by the ants to reflect their experience while optimizing a particular problem.

ACO was initially designed for solving static combinatorial optimization problems [4]. However, the environments of many real-world problems are often dynamic [5]. A dynamic combinatorial optimization problem can be seen as a series of different static problem instances. Hence, a straightforward method to deal with these type of problems is to consider each dynamic change as the arrival of a new problem instance that needs to be solved from scratch. However, this method is often impractical when the dynamic change is relatively small [5], [6], [7]. A better strategy is to adapt to dynamic changes by transferring the past experience of the optimization process since the new environment will be in some sense related to the old one. ACO is a good choice in adapting to dynamic changes because it naturally implements a memory structure (i.e., the pheromone model), allowing ACO to remember the past experience. Therefore, when a dynamic change occurs the past experience can be transferred via the pheromone trails of previously optimized environments [7]. Successful ACO applications to dynamic combinatorial optimization problems include Internet-like network routing [8], vehicle routing [9], and train scheduling [10].

In the last decade, studying ACO algorithms for dynamic environments attracted a lot of attention because of their intrinsic features [6], [7]. Several dynamic strategies have been designed to enhance the adaptation capabilities of ACO [11], [12], [13], [14]. These strategies have been mainly tested on different dynamic variations of the travelling salesperson problem (TSP), which

makes the dynamic TSP (DTSP) an ideal subject for a case study in this article. However, in the existing DTSPs [15], [16], [17], [18], [19], important dynamic optimization settings (e.g., the type, frequency, and magnitude of the dynamic change) are not following any experimentation protocol. This causes many difficulties for researchers to analyze the strengths and weaknesses of algorithms in DTSPs. Consequently, the following important questions arise: *Which ACO algorithm performs best under which dynamic settings and why? Which ACO features are important when addressing dynamic environments?* We strongly believe that it would be beneficial to have a unified dynamic benchmark framework to evaluate algorithms in DTSPs with common dynamic settings [20], [21].

This article aims to provide insights concerning the behavior of ACO using the DTSP as a case study. In order to achieve this aim, we set out the following two objectives. First, the existing ACO algorithms designed for DTSPs are classified according to their framework either as *evaporation-based* or as *population-based*. Second, a unified dynamic benchmark framework is proposed to systematically carry out a critical evaluation of the most important features of ACO algorithms (i.e., the decision rule for constructing solutions, the policy for updating pheromone trails, and the dynamic strategy for enhancing adaptation capabilities) on different test cases. In fact, this benchmark framework will also be able to serve as an initial proving ground for new algorithmic ideas in dynamic environments.

The rest of this article is organized as follows. Section II describes the generation of dynamic test cases utilizing the unified dynamic benchmark framework with the TSP as the base problem. Section III classifies the ACO algorithms from the literature that have been designed for DTSPs. Section IV outlines the experimental setup of our study. Next, Section V presents and analyzes the experimental results. Finally, concluding remarks are presented in Section VI.

II. Dynamic Travelling Salesperson Problem

In this section, we describe how dynamic test cases can be generated from a static problem instance. TSP is used as the base problem to generate the dynamic test cases in this article because it is a problem without too many technicalities (e.g., hard constraints). Hence, it is more convenient to evaluate algorithms because their behaviors will not be obscured by the technicalities of the problem [22]. Additionally, the TSP is an important $\mathcal{NP}$-hard combinatorial optimization problem arising in several applications [23]. Finally, the best ACO algorithms for the TSP very often perform well when applied to more complex problems. For example, the Ant

Colony System [1], which is one of the best performing ACO algorithms on the TSP, is used to solve world-scale instances of vehicle routing problems [24].

A. TSP Formulation

The TSP can be described as follows: given a collection of cities, the objective of a salesperson is to find the shortest Hamiltonian cycle of visiting all of the cities once before finally returning to the starting city. More formally, a TSP problem instance is modeled by a complete directed weighted graph $G = (N, A)$, where N is a set of n nodes and A is a set of arcs fully connecting the nodes. For the classical TSP, nodes and arcs represent respectively the cities and the links between the cities. Each arc $(i, j) \in A$ is associated with a non-negative value $w_{ij} \in \mathbb{R}^+$, which represents the distance between nodes i and j. In this article, we will use symmetric TSP problem instances to generate dynamic test cases and, hence, these distances are independent of the direction of traversing the arcs, that is, $w_{ij} = w_{ji}$ for every pair of nodes.

B. Generating Dynamic Test Environments

Every TSP problem instance consists of a weight matrix that contains all the weights associated with the arcs of the corresponding graph G. In order to generate dynamic test cases the weight matrix of the problem is subject to changes as follows:

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

where $\mathbf{W}(\cdot)$ is the weight matrix and T is the environmental period index which is synchronized with the algorithm during the optimization process. Therefore, the environmental period index is defined as $T = \lceil t/f \rceil$, where f is the frequency of change and t is the evaluation counter of the algorithm.

A particular TSP solution $\pi = [\pi_1, \ldots, \pi_n]$ is a permutation of node indices, and for the DTSP it is evaluated as follows:

$$\phi(\pi, t) = w_{\pi_n \pi_1}(T) + \sum_{i=1}^{n-1} w_{\pi_i \pi_{i+1}}(T). \tag{2}$$

Mainly, there are two components of the graph G representing the problem that can change: 1) the set of nodes [13], and 2) the weights on the arcs [11]. A change to any of these problem components will also cause a change to the weight matrix defined in Equation 1 and, thus, it may affect the algorithm's output: the best output before a change may not be the best (or even feasible) after the change. Real-world applications that encompass the aforementioned types of

dynamic changes can be found in many fields, including transportation. For example, changes in the traffic situation (i.e., weight changes) or changes in the visiting locations (i.e., node changes).

1) DTSP with Node Changes: The key idea to generate a dynamic test case with this type of changes is to replace nodes from the current working node set $N_{in}(T)$, where $N_{in}(0) = N$, with newly introduced nodes drawn from another set $N_{out}(T)$. The latter set $N_{out}(T)$ is initially generated with n new random nodes in the range of the N set. A dynamic change of this type occurs as follows. Every f evaluations exactly $\lceil mn \rceil$ nodes are randomly selected from $N_{out}(T)$ to replace exactly $\lceil mn \rceil$ randomly selected nodes from $N_{in}(T)$, where m ($m \in (0, 1]$) defines the magnitude of change. The higher the value of m, the more nodes will be replaced. In this way, the weight matrix will be affected because the weights on the arcs connecting the replaced nodes will be modified.

2) DTSP with Weight Changes: A dynamic test case with this type of changes can be generated by assigning an increasing/decreasing factor value to the arc connecting nodes i and j as follows:

$$w_{ij}(T+1) = \begin{cases} w_{ij}(0) + \mathcal{R}_{ij}, & \text{if arc } (i,j) \in A_S(T), \\ w_{ij}(T), & \text{otherwise,} \end{cases} \tag{3}$$

where $w_{ij}(0)$ is the initial weight of arc (i, j) (from the static TSP instance when $T = 0$), $\mathcal{R}_{ij}$ is a normally distributed random number (with zero mean and standard deviation set proportional to $w_{ij}(0)$ as in [20]) that defines the modified factor value to arc (i, j), and $A_S(T) \subset A$ defines the set of arcs randomly selected for the change at that period. Consider that the size of the set A is defined by the number of arcs as follows: $n(n-1)$. Then, the size of $A_S(T)$ is defined by the magnitude of change (i.e., $m \in (0, 1]$) and the size of A. Therefore, every f evaluations exactly $\lceil mn(n-1) \rceil$ arcs will be selected to change their weights. The higher the value of m, the more arcs will be selected for changes.

C. Some Additional Remarks

It must be noted that there are some ACO algorithms that benefit from the use of less ants (e.g., the Ant Colony System [1]), resulting in less evaluations per algorithmic iteration, and some other ACO algorithms that benefit from the use of more ants (e.g., the hyper-populated ant colonies [25], [26]), resulting in more evaluations per algorithmic iteration. Therefore, the frequency of change is expressed in evaluations in the described dynamic benchmark framework. In this way, a fair comparison between the competing algorithms is ensured with the generated DTSP test cases because the dynamic changes occur at exactly the same period of time (i.e.,

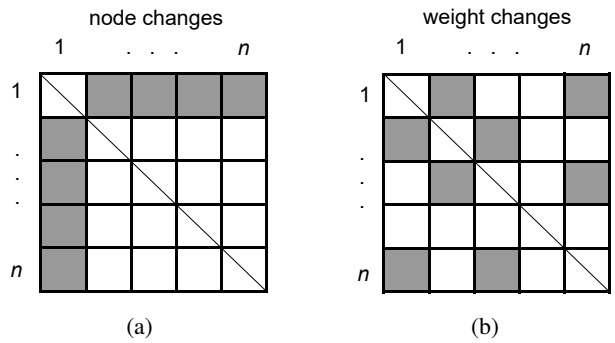


Fig. 1: Illustration of the weight matrix when node (a) and weight (b) dynamic changes (with $m = 0.2$ in this example) occur. Gray boxes denote a change to the weight. Note that the symmetry with respect to the main diagonal line is due to the fact that symmetric problem instances are used to generate DTSPs.

all algorithms have exactly the same number of evaluations available between the environmental changes). Also, the period of change is restricted either at the start or at the end of an algorithmic iteration [27].

From the way the magnitude of change is defined for both DTSP types, there exists an interesting observation: the weights of the same number of arcs, but not necessarily the same arcs, are modified when the same value of m is selected for DTSPs that utilize symmetric problem instances. For example, in Figure 1 the dynamic change with magnitude set to $m = 0.2$ will change one node for a problem of size five (i.e., $\lceil 0.2 \times 5 \rceil$) when node changes occur and four arcs for the same problem (i.e., $\lceil 0.2 \times 5(5-1) \rceil$) when weight changes occur. Consider that there are two arcs connecting a pair of nodes and their weights are the same in symmetric problem instances. For a problem of size five in a fully connected graph, each node is connected with the remaining four nodes; therefore, when one node is replaced, the weights of eight arcs in total will be modified as shown in Figure 1(a). For the same problem, the weights of the four selected arcs will be modified when weight changes occur, but, due to the symmetric property of the problem instance, each time the weight of an arc (i, j) changes, the weight of the arc (j, i) changes to the same value. As a result, the weights of eight arcs in total will actually change (as in the case of node changes) as shown in Figure 1(b).

III. Classification of ACO Algorithms

A. ACO for the Dynamic Travelling Salesperson Problem

In the ACO metaheuristic, a colony of ω artificial ants constructs solutions by incrementally selecting feasible solution components. Each solution component is associated with a pheromone value which is used to guide artificial ants when selecting the next solution component. Since the TSP is considered as the base problem to generate dynamic test cases, it is used as a concrete example to describe the ACO metaheuristic in this section.

With probability $1 - q_0$ ($q_0 \in [0, 1]$) each ant chooses the next node probabilistically. The probability with which ant k chooses node j from node i is defined as follows:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \text{if } j \in \mathcal{N}_i^k, \\ 0, \qquad\qquad \text{otherwise}, \end{cases} \tag{4}$$

while with probability q_0 each ant k chooses the next node j with the highest probability as follows:

$$j = \arg\max_{l \in \mathcal{N}_i^k} \{[\tau_{il}]^\alpha [\eta_{il}]^\beta\}, \tag{5}$$

where τ_{ij} and $\eta_{ij} = 1/w_{ij}(T)$ are, respectively, the pheromone trail value (which is initialized with a value τ_0) and the heuristic value (which is available a priori) of the arc connecting node i to node j, α and β are two parameters that control the relative influence of the pheromone versus the heuristic information, and $\mathcal{N}_i^k$ is the set of nodes that ant k has not selected yet when being at node i.

When $q_0 = 0.0$, we have the normal *random proportional* decision rule in which ants make moves probabilistically using only Equation 4. When $q_0 > 0.0$, we have the so-called *pseudorandom proportional* decision rule, introduced in the Ant Colony System [1], in which ants make the best possible moves as indicated by the existing pheromone trails and the heuristic information using Equation 5, in combination with the probabilistic moves defined in Equation 4.

B. ACO Framework Types

There are two framework types of the ACO metaheuristic. First, the evaporation-based framework that typically utilizes the random proportional decision rule as shown in Algorithm 1. Second, the population-based framework that typically utilizes the pseudorandom proportional decision rule as shown in Algorithm 2. These two frameworks also differ in the way their pheromone trails are updated and, consequently, in the way they adapt to dynamic changes.

Algorithm 1 Evaporation-Based Framework

1: $I \leftarrow 0$ and $t \leftarrow 0$
2: initialize all pheromone trails τ_{ij} uniformly with a value τ_0
3: **while** termination condition is not satisfied **do**
4: construct ω solutions using the random proportional rule in Equation 4
5: evaluate all solutions using Equation 2
6: $t \leftarrow t + \omega$
7: find the best solution π^{ib} from the I–th iteration
8: **if** $\phi(\pi^{ib}, t)$ is better than $\phi(\pi^{bs}, t)$ **then**
9: $\pi^{bs} \leftarrow \pi^{ib}$
10: **end if**
11: reduce all pheromone trails τ_{ij} using Equation 6
12: increase the pheromone trails τ_{ij} using Equation 7
13: $I \leftarrow I + 1$
14: **end while**
15: **OUTPUT**: the best-so-far solution π^{bs}

1) Adapting via Pheromone Evaporation: The evaporation-based framework used in this article adopts the pheromone update policy of the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [28], which is one of the best performing evaporation-based algorithms. The pheromone trails are updated by applying evaporation as follows:

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

where ρ ($\rho \in (0,1]$) is the evaporation rate. After evaporation, the best ant deposits pheromone on the arcs belonging to its constructed solution (i.e., π^{best}) with an amount proportional to the quality of that solution as follows:

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

where $\Delta\tau_{ij}^{best} = 1/\phi(\pi^{best}, t)$ is the proportional amount of pheromone to be deposited. The best ant may be either the best ant found in the current iteration (*iteration-best* ant), in which case $\pi^{best} = \pi^{ib}$, or the best ant found since the start of the algorithm (*best-so-far* ant), in which case $\pi^{best} = \pi^{bs}$, or a combination of both [29]. The lower and upper limits τ_{min} and τ_{max} of the

Algorithm 2 Population-Based Framework

$I \leftarrow 0$ and $t \leftarrow 0$
initialize all pheromone trails τ_{ij} uniformly with a value τ_0
set the population-list $P(I) \leftarrow \emptyset$
while termination condition is not satisfied **do**
  construct ω solutions using the pseudorandom proportional rule in Equations 4 and 5
  evaluate all solutions using Equation 2
  $t \leftarrow t + \omega$
  find the best solution π^{ib} from the I–th iteration
  if population-list $P(I)$ is full **then**
    select the oldest solution π^{out} to leave the population-list $P(I)$
    reduce the pheromone trails τ_{ij} using Equation 9
  end if
  $\pi^{in} \leftarrow \pi^{ib}$
  insert π^{in} in the population-list $P(I)$
  increase the pheromone trails τ_{ij} using Equation 8
  if $\phi(\pi^{ib}, t)$ is better than $\phi(\pi^{bs}, t)$ **then**
    $\pi^{bs} \leftarrow \pi^{ib}$
  end if
  $I \leftarrow I + 1$
end while
OUTPUT: the best-so-far solution π^{bs}

pheromone trail values are imposed such that $\forall(i,j) : \tau_{min} \leq \tau_{ij} \leq \tau_{max}$. The τ_{min} and τ_{max} values are always updated proportionally to the solution quality of the current best-so-far ant.

When a dynamic change occurs, the current pheromone trails will contain mixed information that can either guide the search towards promising areas of the search space containing high-quality solutions or misguide the search towards poor areas of the search space containing low-quality solutions. Hence, some previously generated pheromone trails may lead to a possibly poor solution for the new environment. Therefore, pheromone evaporation is responsible for the gradual reduction of these pheromone trails.

2) Adapting via Population-list: The population-based framework uses a population-list $P(I)$ (i.e., an archive of solutions) of size K, in which at each iteration I the solution of the iteration-best ant is stored in the population-list [12]. The population-list is directly associated with the pheromone trails, that is, whenever an ant in enters the population-list, a positive pheromone update is performed as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau, \forall (i,j) \in \pi^{in}, \tag{8}$$

where $\Delta\tau$ is the constant pheromone amount to be deposited and π^{in} is the solution of the ant entering the population-list (i.e., $\pi^{in} = \pi^{ib}$). Whenever an ant out leaves the population-list, due to the limited size of $P(I)$, a negative pheromone update is performed as follows:

$$\tau_{ij} \leftarrow \tau_{ij} - \Delta\tau, \forall (i,j) \in \pi^{out}, \tag{9}$$

where $\Delta\tau$ is the constant pheromone amount to be removed and π^{out} is the solution of the ant leaving the population-list (i.e., the oldest solution in the population-list). In this way, outdated pheromone trails are reduced directly when a dynamic change occurs.

There exists an interesting relationship between the evaporation-based and population-based frameworks: they both use pheromone trail limits, which is an important feature to avoid early search stagnation [29], although these are explicit, as discussed earlier, in the evaporation-based framework and implicit in the population-based framework. In particular, the pheromone trail values in the population-based framework can never drop below the initial pheromone value τ_0 because a constant amount of pheromone $\Delta\tau$ will always be added to the trails when the ant enters the population-list and the same constant amount of pheromone $\Delta\tau$ will always be removed from the same trails when the ant leaves the population-list. On the contrary, the additional pheromone value a trail can receive can be at most K, which is the size of the population-list, times the constant amount of pheromone $\Delta\tau$. Hence, in the population-based framework is implicitly guaranteed that $\forall (i,j) : \tau_0 \leq \tau_{ij} \leq \tau_0 + K\Delta\tau$.

It must be also noted that the pheromone trail limits (i.e., $\tau_{min} = \tau_0$ and $\tau_{max} = \tau_0 + K\Delta\tau$) in the population-based framework are fixed, whereas the pheromone trail limits in the evaporation-based framework are variable (they change whenever a new best solution is discovered).

C. ACO Variants for the DTSP

As discussed earlier, both ACO frameworks can successfully address DTSPs using their pheromone trails to utilize previous experience and adapt to the newly generated environments,

either via pheromone evaporation in the case of the evaporation-based framework (denoted as $\mathcal{MM}$AS in Table I) or via the population-list in the case of the population-based framework (denoted as P-ACO in Table II). However, these adaptation mechanisms often become ineffective once ACO shows a stagnation behavior (i.e., when all the ants follow the same path and construct the same solution over and over again). This is because excessive growth of pheromone trails will be generated on the arcs of a single solution. Therefore, in case a dynamic change occurs, it will be difficult for the ants to escape from this solution (which could be optimal previously) in order to search for the new optimal solution. Since both frameworks focus their search on a specific area of the search space, there is a consequent danger of getting trapped in a stagnation situation. Although the pheromone trail limit feature, discussed in Section III-B, is designed to counteract this situation, both ACO frameworks need to be further endowed with additional features when addressing dynamic environments [14].

Several dynamic strategies have been designed to balance the knowledge transferred and the diversity maintained in ACO, and, consequently, improve the ACO's adaptation capabilities in dynamic environments [7]. Achieving a good balance between these two factors is not a trivial task. On one hand, increasing the diversity of the constructed solutions resolves the issue of search stagnation. On the other hand, it may disturb the optimization process because of too much randomization. Additionally, transferring knowledge is essential for faster recovery when dynamic changes occur. However, if too much knowledge is transferred or utilized from previous environments, then the reoptimization process may start near a (possibly poor-quality) local optimum and get stuck there. In fact, these two factors are also conflicting because if the diversity maintained is not enough, then it will be difficult for ACO to utilize any knowledge transferred.

Existing ACO variants for the DTSP are classified into evaporation-based and population-based framework variants in Tables I and II, respectively. The main common feature of all these ACO variants is that they prevent the concentration of large amounts of pheromone trails on the arcs of a single solution. In the evaporation-based variants, this is typically achieved by reducing the pheromone trails on these arcs much faster than the remaining pheromone trails and/or increasing some pheromone trails on arcs of other promising solutions in the search space. On the contrary, the population-based variants typically introduce newly generated solutions, that do not belong in the constructing colony, in the population-list. In this way, the population-list is unlikely to maintain identical solutions that will result in stagnation of the search. It must be noted that

TABLE I: Dynamic strategies of evaporation-based variants

ACO Algorithm	Reference	Dynamic Strategy	Explicit Action
$\mathcal{MM}$AS†	[28]	Adapt via pheromone evaporation	–
$Restart$-Strategy	[16]	Reinitialize all pheromone trails by the same degree	Modify pheromone values
τ-Strategy	[16]	Utilize the location of dynamic changes	Modify pheromone values
η-Strategy	[16]	Utilize the location of dynamic changes	Modify pheromone values
$Shake$-Strategy	[11]	Reduce trails with higher pheromone amount more	Modify pheromone values
Max-Strategy	[30]	Reinitialize all trails proportional to the maximum pheromone value	Modify pheromone values
$\mathcal{MM}$AS$_{caste}$	[31]	Multiple colonies with different decision rules	–
$\mathcal{MM}$AS$_R$	[32]	Reinitialize all pheromone trails to the initial value	Modify pheromone values
$\mathcal{MM}$AS$_A$	[33]	Adapt the pheromone evaporation rate	Increase the evaporation rate
$\mathcal{MM}$AS$_S$	[34]	Self-adapt the pheromone evaporation rate	–
MC-$\mathcal{MM}$AS	[35]	Multiple colonies with independent pheromone trails	–

† Conventional evaporation-based framework

TABLE II: Dynamic strategies of population-based variants

ACO Algorithm	Reference	Dynamic Strategy	Explicit Action
P-ACO†	[12]	Adapt via population-list	–
RIACO	[36]	Generate random immigrants	–
EIACO	[36]	Generate elitism-based immigrants	Update previous best solution
HIACO	[36]	Generate both random and elitism-based immigrants	Update previous best solution
M-PACO	[32]	Population-list with triggered random immigrants	–
MIACO	[37]	Generate memory-based immigrants	Update memory structure
EIIACO	[38]	Generate environmental-information based immigrants	Update previous population-list
HIACO-II	[39]	Generate either random or elitism-based immigrants	Update previous best solution

† Conventional population-based framework

the dynamic strategies of most variants are based on explicit actions when a change occurs. For detailed descriptions of these variants, refer to their original references listed in Tables I and II.

IV. Experimental Setup

A. Dynamic Test Cases

TSP instances were obtained from the TSPLIB benchmark library [40], which is available at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, to generate dynamic test cases as described in Section II. Specifically, the frequency of change was set proportionally to the size of the problem instance as follows: $f = 2.5n$ and $f = 25n$, indicating quickly (e.g., before the algorithm converges, denoted as `fast`) and slowly (e.g., after the algorithm has converged, denoted as `slow`) changing environments, respectively. The magnitude of change was set to $m = 0.1$, $m = 0.25$, $m = 0.5$, and $m = 0.75$, indicating small, to medium, to large dynamic changes, respectively. The dynamic settings for each DTSP test case are selected to systematically

analyze the dynamic behavior of ACO algorithms (i.e., their ability to recover fast and produce the best output). Note that usually as the frequency of change is faster and the magnitude of change is increasing the DTSP test case becomes harder to address [14], [21].

B. Parameter Settings

The common parameters of all ACO algorithms were set to typical values as follows [4]: $\alpha = 1$ and $\beta = 5$ (for all the experiments). The colony size ω for DTSPs with node changes was set to $\omega = 5$ and for DTSPs with weight changes was set to $\omega = 25$. The remaining parameters were set to $\rho = 0.8$ and $K = 3$ for ACO algorithms using these parameters.

Note that these parameters were found to yield reasonable performance in most DTSPs. In general, the best results on both types of DTSPs are obtained when setting the evaporation-based variants with a high pheromone evaporation rate and the population-based variants with a small population-list size. On the contrary, the colony size varies for the two types of DTSPs: a smaller colony size performs better in DTSPs with node changes, whereas a larger colony size performs better in DTSPs with weight changes.

For each ACO algorithm on each DTSP test case, 30 independent runs were executed on the same set of random seed numbers. For each run, 100 environmental changes were allowed and the value of the best-so-far ant since the last change of the environment was recorded.

C. Performance Measures

Five metrics for evaluating the performance of ACO algorithms were considered: 1) offline performance, 2) best before change, 3) robustness, 4) diversity, and 5) λ-branching factor. The first two metrics are classified as optimality-based metrics, whereas the last three metrics are classified as behavior-based metrics. The set of metrics is described in the following:

- **Offline performance** ($\bar{P}_{offline}$) measures how well an algorithm performs until it finds a high-quality solution [5]. It is one of the most frequently used metrics for dynamic optimization, which is defined as the average solution quality of the best-so-far solution found at each evaluation step. This metric is calculated as follows:

$$\bar{P}_{offline} = \frac{1}{E} \sum_{t=1}^{E} \phi(\pi^{bs}, t), \tag{10}$$

  where $\phi(\pi^{bs}, t)$ is the quality of the best-so-far solution since the last change of the environment and E is the total number of evaluations (which is calculated as $E = Mf$, where M is the total number of environmental changes and f is the frequency of change).

- **Best before change** ($\bar{P}_{change}$) measures how good the final outcome of an algorithm is [41]. It is another useful metric, defined as the average solution quality of the best-so-far solution found at the end of each environmental period. This metric is calculated as follows:

$$\bar{P}_{change} = \frac{1}{M} \sum_{T=1}^{M} \phi(\pi^{bs}, Tf - 1), \tag{11}$$

where $\phi(\pi^{bs}, Tf-1)$ is the quality of the best-so-far solution achieved just before the T–th environmental change.

- **Robustness** ($\bar{P}_{robust}$) measures the stability of an algorithm when dynamic changes occur [32], [42]. Suppose that at evaluation count t a change occurs. Then, the robustness of an algorithm is calculated by comparing the quality of the best-so-far solution just before the change (i.e., at evaluation count $t-1$) with the quality of the best-so-far solution when a change occurs (i.e., at evaluation count t) as follows:

$$\bar{P}_{robust} = \frac{1}{M-1} \sum_{T=1}^{M-1} \begin{cases} 1, & \text{if } \frac{\phi(\pi^{bs}, Tf-1)}{\phi(\pi^{bs}, Tf)} > 1, \\ \frac{\phi(\pi^{bs}, Tf-1)}{\phi(\pi^{bs}, Tf)}, & \text{otherwise.} \end{cases} \tag{12}$$

The values of $\bar{P}_{robust}$ are normalized and range over the interval $[0, 1]$, where a value closer to 1 indicates better average robustness because the quality of the best-so-far solution will have less degradation at each environmental change.

- **Diversity** ($\bar{\mathcal{D}}$) gives an indication of the amount of exploration an algorithm performs [32], which, in this case study, is calculated as follows:

$$\bar{\mathcal{D}} = \frac{1}{\omega(\omega-1)} \sum_{p=1}^{\omega} \sum_{q \neq p}^{\omega} \left(1 - \frac{CA(p,q)}{n}\right), \tag{13}$$

where $CA(p,q)$ is the total number of common arcs between the solutions constructed by the p–th and q–th ants. The $\bar{\mathcal{D}}$ value tends to decrease as the ants start to follow similar paths, where a value of zero corresponds to stagnation of the search.

- **λ-Branching factor** ($\bar{\lambda}$) measures the distribution of the pheromone trail values [43]. The λ-branching factor is given by the number of arcs incident to node i satisfying the following condition: $\tau_{ij} \geq \tau^i_{min} + \lambda(\tau^i_{max} - \tau^i_{min})$, where τ^i_{max} and τ^i_{min} are, respectively, the maximum and minimum pheromone values on arcs incident to node i, and $\lambda \in [0, 1]$ is a constant. The average λ-branching factor (i.e., $\bar{\lambda}$) from all nodes' λ-branching factors gives an indication of the level of search space exploration generated by the ants. The values of $\bar{\lambda}$ range over the interval $[2, n-1]$, where a value of 2 indicates stagnation behavior.

Finally, in order to support our comparisons, pairwise Wilcoxon rank sum statistical tests with a significance level of 0.05 were performed. In the case of multiple comparisons, Kruskal–Wallis statistical tests were performed, followed by posthoc pairwise comparisons using Wilcoxon rank sum statistical tests with p–values adjusted by Bonferroni correction.

V. Experimental Results and Their Analysis

This section presents representative examples of the results obtained in our experimental studies and analyzes them[1].

A. Comparison between Evaporation-Based and Population-Based Frameworks

In this study, the evaporation-based framework is compared with the population-based framework. The experiments were carried out using three TSPLIB instances to generate the dynamic test cases: `kroA200`, `rd400` and `u1060`, in which the number in the instance name identifies the problem size, and they are considered as small, medium and large instances, respectively. Table III shows the $\bar{P}_{offline}$, $\bar{P}_{change}$, and $\bar{P}_{robust}$ results obtained by the two frameworks for DTSPs with $m = 0.25$. To better understand their behavior, Figure 2 shows plots of the $\bar{\lambda}$ (with $\lambda = 0.05$) and $\bar{\mathcal{D}}$ results against the last ten node changes on the TSPLIB instance `rd400`. From the experimental results, the following observations can be drawn.

First, in terms of $\bar{P}_{offline}$, the population-based framework performs significantly better than the evaporation-based framework in most quickly changing DTSPs. This is because the pseudo-random proportional decision rule used in the population-based framework is more greedy than the random proportional decision rule used in the evaporation-based framework and, thus, it directs the search towards high-quality solutions quickly. On the contrary, the random proportional decision rule requires sufficient time to explore the search space (and most probably discover better-quality solutions). Hence, the evaporation-based framework performs significantly better, in terms of both $\bar{P}_{offline}$ and $\bar{P}_{change}$, than the population-based framework in most slowly changing DTSPs. Another reason is that the evaporation-based framework requires some time to make significant changes to the pheromone trails, because the pheromone evaporation will gradually decrease outdated pheromone trails down to the explicit τ_{min} value, whereas the population-based framework decreases outdated pheromone trails much faster, because the pheromone on

[1]The detailed results of each experimental study are provided as supplementary material. The supplementary material is available at https://github.com/Mavrovouniotis/ACODTSP.

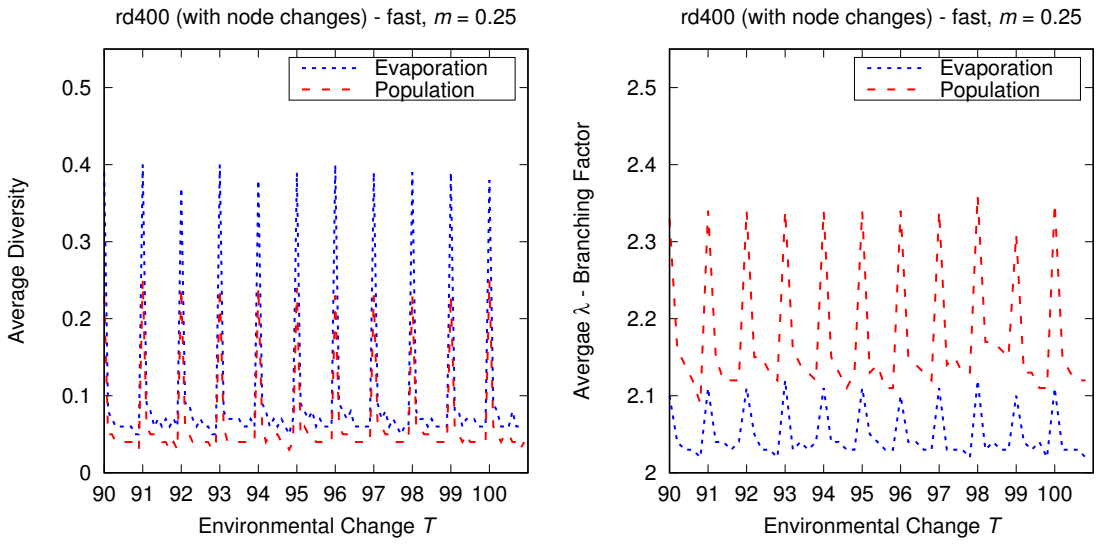
TABLE III: $\bar{P}_{offline}$, $\bar{P}_{change}$, and $\bar{P}_{robust}$ (averaged over 30 runs) results of evaporation-based and population-based frameworks (i.e., $\mathcal{MM}$AS and P-ACO, respectively) for DTSPs with $m = 0.25$.

Metric	ACO Framework	kroA200		rd400		u1060	
		weights	nodes	weights	nodes	weights	nodes
		fast					
$\bar{P}_{offline}$	Evaporation	**30140**	34620	16361	17322	254703	325217
	Population	30552	**34430**	**16276**	**17203**	**252955**	**321851**
$\bar{P}_{change}$	Evaporation	**29620**	**33021**	15992	16602	249219	314794
	Population	30101	33118	15983	16594	**247801**	**312261**
$\bar{P}_{robust}$	Evaporation	0.95	0.77	0.93	0.76	0.92	0.76
	Population	**0.96**	**0.82**	**0.95**	**0.82**	0.92	**0.82**
		slow					
$\bar{P}_{offline}$	Evaporation	**28300**	**31635**	**14947**	**15892**	241128	306628
	Population	28745	31887	15154	15952	**239814**	**304820**
$\bar{P}_{change}$	Evaporation	**27964**	**30771**	**14641**	**15403**	236858	300317
	Population	28363	31113	14870	15527	**235303**	**299506**
$\bar{P}_{robust}$	Evaporation	0.94	0.72	0.93	0.71	0.91	0.73
	Population	0.94	**0.78**	0.93	**0.77**	0.91	**0.79**

Bold values indicate statistical significance.

the arcs belonging to the solutions to be removed from the population-list will be directly set to the τ_0 value (which is also the implicit τ_{min} value). Nevertheless, it appears that the time required for the random proportional decision rule to express its effect is affected by the scale of the problem. For example, on the TSPLIB instance u1060, the population-based framework significantly outperforms the evaporation-based framework even in slowly changing DTSPs. This is natural because in large problem instances with huge search spaces the random proportional decision rule will spend an enormous amount of exploration time until it discovers areas in the search space containing high-quality solutions.

Second, the population-based framework usually maintains a higher λ-branching factor than the evaporation-based framework [see Figure 2(b)]. This is because the pheromone trails of the evaporation-based framework are more likely to represent a single solution (i.e., the solution of the best ant). In particular, the pheromone on the arcs corresponding to this solution will rise up to the explicit τ_{max} value, while on all the other arcs the pheromone will decrease down to the explicit τ_{min} value. On the contrary, the pheromone trails of the population-based framework, are more likely to represent multiple solutions (i.e., the solutions stored in the population-list).



(a) (b)

Fig. 2: $\bar{\mathcal{D}}$ (a) and $\bar{\lambda}$ (b) (averaged over 30 runs) results of evaporation-based and population-based frameworks against the last ten environmental changes.

Hence, the pheromone on the arcs corresponding to these solutions will increase (and most probably have the same value), while on all the other arcs the pheromone will remain equal to the implicit τ_{min} value.

Third, in terms of $\bar{P}_{robust}$, the population-based framework significantly outperforms the other framework in almost all DTSPs. Nevertheless, the robustness of both frameworks is very high for DTSPs with weight changes. This is because the environmental changes generated by the weight changes may not affect the current best-so-far solution (the arcs in which their weights change may not necessarily belong to that solution). On the contrary, if a dynamic change occurs to the nodes, it will always affect the current best-so-far solution and, consequently, affect the current optimization process. Recall from Figure 1 that all arcs incident to the affected node will be modified. Therefore, it is guaranteed that at least one affected arc will belong to the current best-so-far solution.

Fourth, it is interesting to observe that although the average λ-branching factor of the population-based framework suggests more exploration than the evaporation-based framework [see Figure 2(b)], the average diversity the former framework generates is usually lower than the latter framework [see Figure 2(a)]. This is because the pseudorandom proportional decision rule

exploits the search experience accumulated by the colony more strongly than the random proportional decision rule as discussed previously. Consequently, the solutions constructed will have a relatively large number of common arcs, resulting in lower $\bar{\mathcal{D}}$ results.

B. Effect of Main Framework Features

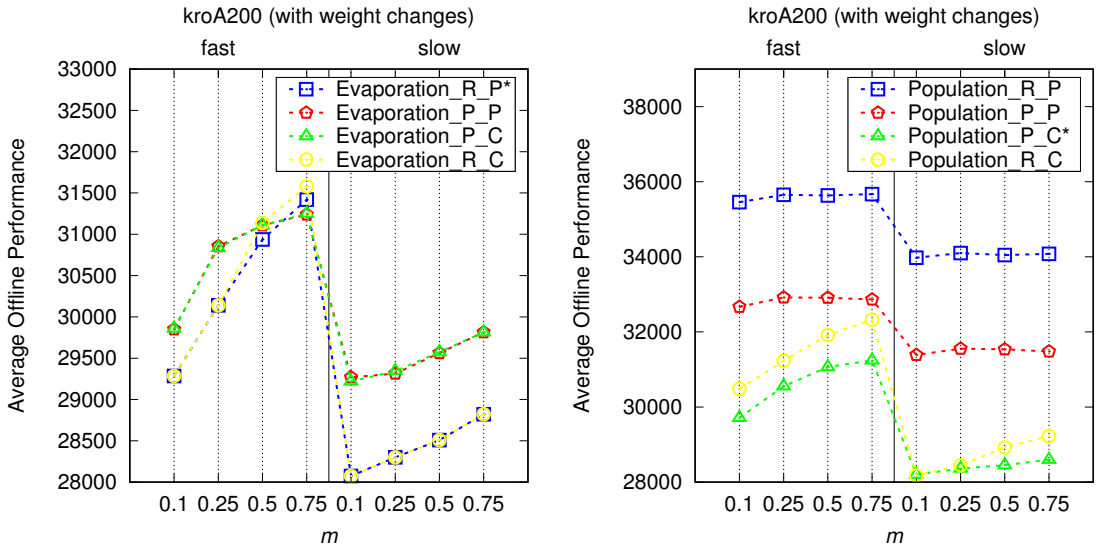
In this study, the two main features of the evaporation-based and population-based frameworks are investigated: 1) decision rule (i.e., random denoted as `R` or pseudorandom denoted as `P`), and 2) pheromone update policy (i.e., proportional denoted as `P` or constant denoted as `C`). Specifically, the two ACO frameworks with their default decision rule and pheromone update policy are compared with alternative feature combinations (in the format "ACO framework"_"`decision rule`"_"`pheromone policy`"). For example, the default feature combinations of the evaporation-based and population-based frameworks are denoted as Evaporation_`R_P` and Population_`P_C`, respectively. The experiments were carried out using the same set of TSPLIB instances as in the previous study. Figure 3 shows the $\bar{P}_{offline}$ results obtained by the two frameworks with all possible feature combinations on the TSPLIB instance `kroA200`.

From Figure 3(a) it can be observed that the constant amount of pheromone has no effect on the performance of the evaporation-based framework. This is because only a single ant is allowed to add pheromones in each iteration. Also, from Figure 3(b) it can be observed that the proportional amount of pheromone degrades the performance of the population-based framework. This is because the resulting amount of pheromone representing the population-list solutions is very small to bias the search.

Furthermore, the use of the random proportional decision rule has a negative effect on the performance of the population-based framework. As we have seen previously, the way pheromones are distributed on the arcs of the constructing graph promote search space exploration. This is reflected by a higher λ-branching factor for the population-based framework [see Figure 2(b)]. Therefore, a decision rule that further promotes the exploration may result in too much randomization. Similarly, the use of the pseudorandom proportional decision rule has a negative effect on the performance of the evaporation-based framework. This shows that the two main ACO framework features often complement each other.

C. Effect of Dynamic Strategies

In this study, the dynamic strategies integrated with the evaporation-based and population-based frameworks are investigated. The experiments were carried out using a wider set of TSPLIB



(a) (b)

Fig. 3: $\bar{P}_{offline}$ (averaged over 30 runs) results of evaporation-based (a) and population-based (b) frameworks with alternative decision rules and pheromone update policies. *These combinations are the default ones.

instances than in the previous studies. From the statistical comparisons of the evaporation-based and population-based frameworks against their variants, listed in Tables I and II, respectively, the following observations can be highlighted.

First, evaporation-based variants performing severe explicit actions, such as $\mathcal{MM}\text{AS}_R$ and Max-Strategy variants, are not effective for DTSPs with weight changes. As discussed earlier, the environmental changes generated by weight changes usually affect the current best-so-far solution less than when the environmental changes are generated by node changes. Hence, a reinitialization of the pheromone trails (either to the initial pheromone value with $\mathcal{MM}\text{AS}_R$ or to the maximum pheromone value with Max-Strategy) may destroy useful previous knowledge. On the contrary, these variants perform significantly better than the evaporation-based framework in most DTSPs with node changes (except when $m = 0.1$, in which the changing environments are more likely to be similar).

Second, MC-$\mathcal{MM}$AS and $\mathcal{MM}\text{AS}_{caste}$ variants perform significantly better than the evaporation-based framework in several DTSPs with node changes, while they maintain a competitive performance in DTSPs with weight changes. These variants focus their search on multiple areas

of the search space (either using multiple colonies with MC-$\mathcal{MM}$AS or multiple castes with $\mathcal{MM}\text{AS}_{caste}$). Therefore, the past experience available to be utilized is broadened when a change occurs.

Third, EIACO, MIACO and HIACO-II variants perform significantly better than the population-based framework in most DTSPs, both with node and weight changes. It is interesting to note that these variants have a common feature: their dynamic strategy generates elitism-based immigrants. This shows that this dynamic strategy is responsible for the performance improvement. Elitism-based immigrants are generated via transferring knowledge from the previous environment and replace solutions in the population-list to maintain diversity. Therefore, when the changing environments are similar (e.g., when $m = 0.1$), the utilization of this knowledge will guide the search process to promising areas faster.

Fourth, RIACO and EIIACO variants are effective only for a few extreme DTSP test cases (i.e., when f is `fast` and $m = 0.75$). Otherwise, they perform significantly worse than the population-based framework in most DTSPs. This is due to the fact that the diversity generated is very high and therefore the ongoing optimization process is disturbed.

D. Effect of Utilizing Change-Related Information

From a practical perspective, change-related information may be available in real time to the optimizer (e.g., using advances in information and communication technologies or related technologies, in the case of transportation systems). In this study, the outdated solutions are repaired using change-related information: the affected nodes are removed and the newly introduced nodes are placed in the best possible position when a change occurs [12]. The experiments were carried out on the same set of TSPLIB instances as in the previous study. Figure 4 shows the $\bar{P}_{offline}$ results obtained by the evaporation-based and population-based frameworks when change-related information is utilized and when it is not, on the TSPLIB instance `kroA150`. In addition, these results are compared with the results obtained by the $Restart$-Strategy, η-Strategy, and τ-Strategy variants that were specifically designed to utilize change-related information as listed in Table I.

From Figure 4, the following observations can be drawn. First, the performance of both ACO frameworks is significantly improved when change-related information is utilized in all DTSPs. Second, the three competing strategies outperform the two ACO frameworks only when they

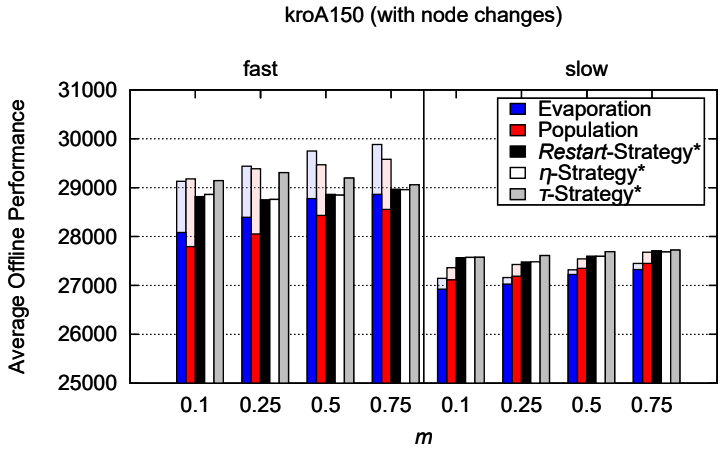


Fig. 4: $\bar{P}_{offline}$ (averaged over 30 runs) results of evaporation-based and population-based frameworks, and three evaporation-based variants. Each bar is divided into two parts that represent the results when utilizing change-related information (darker) or not (lighter). *These variants have been designed to utilize change-related information and, thus, the values when information is not utilized do not exist.

do not utilize change-related information. This shows the effectiveness of utilizing such kind of information to explore and/or exploit the affected areas in the search space faster.

E. Comparisons with Evolutionary Algorithms

In this study, the two ACO frameworks (i.e., $\mathcal{MM}$AS and P-ACO) and two of the best performing variants (i.e., MC-$\mathcal{MM}$AS and EIACO) are compared with a state-of-the-art evolutionary algorithm that utilizes one of the best performing TSP search operators (i.e., the generalized partitioned crossover (GPX) [44]) and the elitism-based immigrant genetic algorithm (EIGA), which is one of the best performing algorithms in evolutionary dynamic optimization [20], [45], and also the evolutionary counterpart of EIACO.

Since we compare algorithms which are structurally different, the frequency of change is set to $f = 100n$ (which is considered as `slow`) to allow sufficient evaluations between the dynamic changes for all types of algorithms. The magnitude of change is randomly chosen from a uniform distribution in $(0.0, 0.5]$ (which is considered small to medium) for every dynamic change. The experiments were carried out on a different set of TSPLIB instances from the previous studies. Table IV shows the $\bar{P}_{offline}$ results obtained by the aforementioned algorithms for DTSPs with weight changes.

TABLE IV: $\bar{P}_{offline}$ results (averaged over 30 runs) of ACO algorithms compared with evolutionary algorithms for DTSPs with weight changes.

TSPLIB Instance	P-ACO	$\mathcal{MM}$AS	EIGA	GPX	EIACO	MC-$\mathcal{MM}$AS
berlin52	7261	7195	7414	7392	**7177**	<u>7191</u>
eil101	572	<u>568</u>	581	578	<u>569</u>	**567**
kroB200	28481	<u>28261</u>	29161	29026	**28231**	<u>28345</u>
lin318	40154	<u>39957</u>	41543	41054	40456	**39932**
pr439	105376	104591	105904	106193	104633	**103918**
p654	49138	49415	48178	**47921**	49127	49533
rat783	**8434**	8521	8515	8509	<u>8436</u>	<u>8444</u>
pr1002	270701	274370	281952	279321	**268532**	274301
u1432	158822	160881	163427	161203	**157503**	161244

Bold values indicate statistical significance.
<u>Underline</u> values indicate no statistical difference with the bold value.

From Table IV, it can be observed that the two ACO frameworks outperform the evolutionary algorithms in almost all DTSPs. This is because for this class of problems (i.e., with a network environment) the pheromone structure of ACO algorithms is built across the solution search space as a weighted graph and it can be taken as the natural representation of past environmental information. Therefore, there is a larger capacity of information that ACO can utilize from its pheromone structure when a dynamic change occurs in comparison to evolutionary algorithms (which are restricted only to the information of their evolving populations).

Also, it is interesting to observe that EIACO outperforms EIGA although both algorithms utilize the same dynamic strategy. Once again, it can be observed that: 1) EIACO outperforms P-ACO in most DTSPs, 2) MC-$\mathcal{MM}$AS is competitive with $\mathcal{MM}$AS, and 3) P-ACO outperforms $\mathcal{MM}$AS as the size of the problem increases. These comparisons are consistent with the observations found in our previous studies.

VI. Conclusions

This article investigates the behavior of the ACO metaheuristic in dynamic environments. Several ACO algorithms are classified according to their framework (i.e., evaporation-based and population-based). The travelling salesperson problem is used as the base problem to generate dynamic test cases using a proposed dynamic benchmark framework. Experimental studies were systematically conducted to investigate the effect of different features on the performance of ACO in dynamic environments.

From the experimental results, the following concluding remarks can be drawn. First, the effect on the performance of the decision rule used to construct solutions strongly depends on the pheromone update policy of the ACO framework. Second, the dynamic strategies further enhance the adaptation capabilities of the two ACO frameworks, but their effect on the performance strongly depends on the dynamic settings of the problem. And third, the utilization of change-related information is always effective for both ACO frameworks.

Finally, the source code of the dynamic benchmark framework together with the performance measures and the ACO algorithms used in this case study is available at https://github.com/Mavrovouniotis/ACODTSP. The source code can be useful to researchers who are interested in generating the same or different dynamic test cases to compare their own algorithms (not necessarily ACO) in dynamic environments.

ACKNOWLEDGMENTS

This work was supported in part 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, and in part by the National Natural Science Foundation of China under Grants 61673331 and 61673355.

REFERENCES

[1] 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.

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

[3] S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels, "Self-organized shortcuts in the Argentine ant," *Naturwissenschaften*, vol. 76, no. 12, pp. 579–581, Dec. 1989.

[4] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006.

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

[6] G. Leguizamón and E. Alba, "Ant colony based algorithms for dynamic optimization problems," in *Metaheuristics for Dynamic Optimization*, E. Alba, A. Nakib, and P. Siarry, Eds. Berlin, Heidelberg: Springer, 2013, pp. 189–210.

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

[8] G. Di Caro and M. Dorigo, "AntNet: Distributed stigmergetic control for communications networks," *J. Artif. Intell. Res.*, vol. 9, pp. 317–365, Dec. 1998.

[9] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati, "Ant colony system for a dynamic vehicle routing problem," *J. Combinatorial Optimization*, vol. 10, no. 4, pp. 327–343, Dec. 2005.

[10] J. Eaton, S. Yang, and M. Mavrovouniotis, "Ant colony optimization with immigrants schemes for the dynamic railway junction rescheduling problem with multiple delays," *Soft Comput.*, vol. 20, no. 8, pp. 2951–2966, Aug. 2016.

[11] C. Eyckelhof and M. Snoek, "Ant systems for a dynamic TSP," in *Ant Algorithms*, LNCS, vol. 2463, M. Dorigo, G. Di Caro, and M. Sampels, Eds. Berlin, Heidelberg: Springer, 2002, pp. 88–99.

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

[13] M. Guntsch, M. Middendorf, and H. Schmeck, "An ant colony optimization approach to dynamic TSP," in *Proc. 3rd Annu. Conf. Genetic and Evolutionary Computation, GECCO'01*, San Francisco, CA, USA: Morgan Kaufmann, 2001, pp. 860–867.

[14] 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, Oct. 2013.

[15] J. Liu, "Rank-based ant colony optimization applied to dynamic traveling salesman problems," *Eng. Optimization*, vol. 37, no. 8, pp. 831–847, 2005.

[16] M. Guntsch and M. Middendorf, "Pheromone modification strategies for ant algorithms applied to dynamic TSP," in *Applications of Evolutionary Computation*, LNCS, vol. 2037, E. J. W. Boers, Ed. Berlin, Heidelberg: Springer, 2001, pp. 213–222.

[17] C. Silva and T. Runkler, "Ant colony optimization for dynamic traveling salesman problems," in *ARCS Workshops*, 2004, pp. 259–266.

[18] L. Kang, A. Zhou, B. McKay, Y. Li, and Z. Kang, "Benchmarking algorithms for dynamic travelling salesman problems," in *Proc. 2004 IEEE Congr. Evolutionary Computation*, Portland, OR, 2004, pp. 1286–1292.

[19] A. Siemiński, "Using ACS for dynamic traveling salesman problem," in *New Research in Multimedia and Internet Systems*, vol. 314, A. Zgrzywa, K. Choroś, and A. Siemiński, Eds. Cham: Springer, 2015, pp. 145–155.

[20] R. Tinós, D. Whitley, and A. Howe, "Use of explicit memory in the dynamic traveling salesman problem," in *Proc. 2014 Annu. Conf. Genetic and Evolutionary Computation, GECCO'14*, New York, NY: ACM, 2014, pp. 999–1006.

[21] M. Mavrovouniotis, S. Yang, and X. Yao, "A benchmark generator for dynamic permutation-encoded problems," in *Parallel Problem Solving from Nature, PPSN XII*, LNCS, vol. 7492, C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, Eds. Berlin, Heidelberg: Springer, 2012, pp. 508–517.

[22] T. Weise, R. Chiong, J. Lassig, K. Tang, S. Tsutsui, W. Chen, Z. Michalewicz, and X. Yao, "Benchmarking optimization algorithms: An open source framework for the traveling salesman problem," *IEEE Comput. Intell. Mag.*, vol. 9, no. 3, pp. 40–52, Aug. 2014.

[23] H. Handa, L. Chapman, and X. Yao, "Robust route optimization for gritting/salting trucks: A CERCIA experience," *IEEE Comput. Intell. Mag.*, vol. 1, no. 1, pp. 6–9, Feb. 2006.

[24] A. E. Rizzoli, R. Montemanni, E. Lucibello, and L. M. Gambardella, "Ant colony optimization for real-world vehicle routing problems," *Swarm Intell.*, vol. 1, no. 2, pp. 135–151, Dec. 2007.

[25] A. Siemiński, "Using hyper populated ant colonies for solving the TSP," *Vietnam J. Comput. Sci.*, vol. 3, no. 2, pp. 103–117, May 2016.

[26] A. Prakasam and N. Savarimuthu, "Novel local restart strategies with hyper-populated ant colonies for dynamic optimization problems," *Neural Comput. Appl.*, vol. 31, no. 1, pp. 63–76, Jan. 2019.

[27] M. Mavrovouniotis, F. M. Müller, and S. Yang, "Ant colony optimization with local search for dynamic travelling salesman problems," *IEEE Trans. Cybern.*, vol. 47, no. 7, pp. 1743–1756, July 2017.

[28] T. Stützle and H. Hoos, "$\mathcal{MAX}$–$\mathcal{MIN}$ ant system and local search for the traveling salesman problem," in *Proc. 1997 IEEE Int. Conf. Evol. Comput.*, Indianapolis, IN, 1997, pp. 309–314.

[29] T. Stützle and H. H. Hoos, "$\mathcal{MAX}$–$\mathcal{MIN}$ ant system," *Future Gener. Comput. Syst.*, vol. 16, no. 8, pp. 889–914, June 2000.

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

[31] 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, vol. 7824, M. Tomassini, A. Antonioni, F. Daolio, and P. Buesser, Eds. Berlin, Heidelberg: Springer, 2013, pp. 179–188.

[32] 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, July 2011.

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

[34] M. Mavrovouniotis and S. Yang, "Ant colony optimization with self-adaptive evaporation rate in dynamic environments," in *2014 IEEE Symp. Computational Intelligence Dynamic and Uncertain Environments*, Orlando, FL, USA, 2014, pp. 47–54.

[35] M. Mavrovouniotis and S. Yang, "Multi-colony ant algorithms for the dynamic travelling salesman problem," in *2014 IEEE Symp. Computational Intelligence Dynamic and Uncertain Environments*, Orlando, FL, USA, 2014, pp. 9–16.

[36] M. Mavrovouniotis and S. Yang, "Ant colony optimization with immigrants schemes for dynamic environments," in *Parallel Problem Solving from Nature, PPSN XI*, LNCS, vol. 6239, R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, Eds. Berlin, Heidelberg: Springer, 2010, pp. 371–380.

[37] M. Mavrovouniotis and S. Yang, "Memory-based immigrants for ant colony optimization in changing environments," in *Applications of Evolutionary Computation*, LNCS, vol. 6624, C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, and G. Yannakakis, Eds. Berlin, Heidelberg: Springer, 2011, pp. 324–333.

[38] M. Mavrovouniotis and S. Yang, "An immigrants scheme based on environmental information for ant colony optimization for the dynamic travelling salesman problem," in *Artificial Evolution*, LNCS, vol. 7401, J.-K. Hao, P. Legrand, P. Collet, N. Monmarché, E. Lutton, and M. Schoenauer, Eds. Berlin, Heidelberg: Springer, 2012, pp. 1–12.

[39] M. Mavrovouniotis and S. Yang, "Interactive and non-interactive hybrid immigrants schemes for ant algorithms in dynamic environments," in *Proc. 2014 IEEE Congr. Evolutionary Computation*, Beijing, China, 2014, pp. 1542–1549.

[40] G. Reinelt, "TSPLIB–A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, Nov. 1991.

[41] K. Trojanowski and Z. Michalewicz, "Searching for optima in non-stationary environments," in *Proc. 1999 IEEE Congr. Evolutionary Computation*, Washington, DC, USA, 1999, pp. 1843–1850.

[42] Y. Jin, K. Tang, X. Yu, B. Sendhoff, and X. Yao, "A framework for finding robust optimal solutions over time," *Memetic Computing*, vol. 5, no. 1, pp. 3–18, Mar. 2013.

[43] L. M. Gambardella and M. Dorigo, "Ant-Q: A reinforcement learning approach to the traveling salesman problem," in *Proc. 12th Int. Conf. Machine Learning*, A. Prieditis and S. Russell, Eds. San Francisco, CA, USA: Morgan Kaufmann, 1995, pp. 252–260.

[44] D. Whitley, D. Hains, and A. Howe, "Tunneling between optima: Partition crossover for the traveling salesman problem," in *Proc. 11th Annu. Conf. Genetic and Evolutionary Computation, GECCO'09*, New York, NY: ACM, 2009, pp. 915–922.

[45] S. Yang, "Genetic algorithms with memory- and elitism-based immigrants in dynamic environments", *Evol. Comput.*, vol. 16, no. 3, pp. 385–416, 2008.

Ant Colony Optimization Algorithms for Dynamic Optimization: A Case Study of the Dynamic Travelling Salesperson Problem – Supplementary Material

Michalis Mavrovouniotis, Shengxiang Yang, Mien Van, Changhe Li, and Marios Polycarpou

IV. Parameter Settings

The common parameters of all ACO algorithms used were set to typical values (i.e., $\alpha = 1$ and $\beta = 5$) for all the experiments. The colony size ω for each framework was investigated for the two types of DTSPs separately with values $\omega = \{50, 25, 10, 5\}$. In addition, the key parameter of the evaporation-based framework variants (i.e., the evaporation rate ρ) was investigated with values $\rho = \{0.1, 0.2, 0.5, 0.8\}$ and the key parameter of the population-based framework variants (i.e., the population-list size pop) was investigated with values $pop = \{2, 3, 5, 10\}$. The replacement ratio r_i of the generated immigrants for RIACO, EIACO, HIACO, HIACO-II, MIACO and EIIACO was investigated with values $r_i = \{0.1, 0.5, 0.8\}$. For $\mathcal{MM}\text{AS}_S$ the number of discrete rate values available to the self-adaptive evaporation mechanism was investigated with values ranging from 5 to 50. For $\mathcal{MM}\text{AS}_{caste}$, one caste uses the random proportional decision rule while the other uses the pseudorandom proportional decision rule, and MC-$\mathcal{MM}$AS uses two independent colonies.

The combination of these parameters that were found to yield reasonable performance is $\omega = 5$ for most DTSPs with node changes and $\omega = 25$ for most DTSPs with weight changes for all ACO algorithms. Furthermore, for both DTSPs with node and weight changes the remaining parameters are $\rho = 0.8$, $pop = 3$, $r_i = 0.5$ and 20 discrete rate values for ACO algorithms using these parameters.

Michalis Mavrovouniotis, KIOS Research and Innovation Center of Excellence, Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus

Shengxiang Yang, School of Computer Science and Informatics, De Montfort University, Leicester, UK

Mien Van, College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, UK

Changhe Li, School of Automation and Hubei Key Laboratory of Advanced Control and Intelligent Automation for Complex Systems, China University of Geosciences, Wuhan, China

Marios Polycarpou, KIOS Research and Innovation Center of Excellence, Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus

V. Experimental Results and Their Analysis

A. Comparison Between Evaporation-Based and Population-Based Frameworks

TABLE IV: Experimental results regarding $\bar{P}_{offline}$, $\bar{P}_{change}$ and $\bar{P}_{robust}$ (averaged over 30 runs) of evaporation-based and population-based frameworks for DTSPs.

Metric	ACO Framework	kroA200				rd400				u1060			
DTSPs with Weight Changes													
	fast, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
$\bar{P}_{offline}$	Evaporation	**29285**	**30140**	**30938**	31420	15798	16361	16664	16752	251389	254703	257307	258630
	Population	29712	30552	31064	**31240**	15801	**16276**	**16541**	**16609**	**250003**	**252955**	**255351**	**256632**
$\bar{P}_{change}$	Evaporation	**29033**	**29620**	**30142**	30479	15609	15992	16177	16209	247975	249219	250328	250979
	Population	29479	30101	30419	30535	15644	15983	**16151**	16195	**246670**	**247801**	**249051**	**249767**
$\bar{P}_{robust}$	Evaporation	0.98	0.95	0.93	0.92	0.97	0.93	0.91	0.89	0.96	0.92	0.89	0.88
	Population	0.98	**0.96**	**0.94**	**0.93**	0.97	**0.95**	**0.92**	**0.92**	0.96	0.92	**0.90**	**0.89**
	slow, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
$\bar{P}_{offline}$	Evaporation	**28075**	**28300**	**28509**	**28821**	**14600**	**14947**	**15234**	**15309**	240124	241128	242023	242629
	Population	28400	28745	29052	29304	14803	15154	15393	15477	**238941**	**239814**	**240815**	**241573**
$\bar{P}_{change}$	Evaporation	**27897**	**27964**	**27954**	**28116**	**14487**	**14641**	**14729**	**14758**	237205	236858	236861	237100
	Population	28200	28363	28450	28596	14678	14870	14974	15014	**235937**	**235303**	**235368**	**235786**
$\bar{P}_{robust}$	Evaporation	0.97	0.94	0.90	0.88	0.97	0.93	0.88	0.85	0.95	0.91	0.86	0.84
	Population	0.97	0.94	**0.91**	**0.89**	0.97	0.93	**0.89**	**0.87**	0.95	0.91	**0.87**	**0.85**
DTSPs with Node Changes													
	fast, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
$\bar{P}_{offline}$	Evaporation	33923	34620	34751	34787	17208	17322	17414	17391	321956	325217	326427	326616
	Population	**33787**	**34430**	**34373**	**34189**	**17116**	**17203**	**17186**	**17099**	**319221**	**321851**	**321462**	**320080**
$\bar{P}_{change}$	Evaporation	**32599**	**33021**	**33032**	33093	16598	16602	16657	16650	313117	314794	315653	315710
	Population	32651	33118	33091	33000	**16571**	16594	**16613**	**16576**	**310786**	**312261**	**312616**	**312250**
$\bar{P}_{robust}$	Evaporation	0.80	0.77	0.75	0.76	0.79	0.76	0.76	0.76	0.79	0.76	0.76	0.77
	Population	**0.84**	**0.82**	**0.83**	**0.84**	**0.83**	**0.82**	**0.83**	**0.85**	**0.83**	**0.82**	**0.84**	**0.86**
	slow, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
$\bar{P}_{offline}$	Evaporation	**31186**	**31635**	**31621**	**31654**	**15897**	**15892**	**15952**	**15933**	305124	306628	307077	307743
	Population	31416	31887	31855	31782	15943	15952	15982	15957	**303528**	**304820**	**305068**	**304999**
$\bar{P}_{change}$	Evaporation	**30498**	**30771**	**30726**	**30736**	**15469**	**15403**	**15446**	**15416**	300109	300317	300536	301261
	Population	30796	31113	31053	30993	15575	15527	15554	15533	**298819**	**299506**	**299983**	**299952**
$\bar{P}_{robust}$	Evaporation	0.76	0.72	0.70	0.70	0.75	0.71	0.70	0.70	0.74	0.73	0.73	0.74
	Population	**0.80**	**0.78**	**0.78**	**0.79**	**0.79**	**0.77**	**0.78**	**0.79**	**0.80**	**0.79**	**0.80**	**0.82**

Bold values indicate statistical significance

B. *Effect of Main Framework Features*

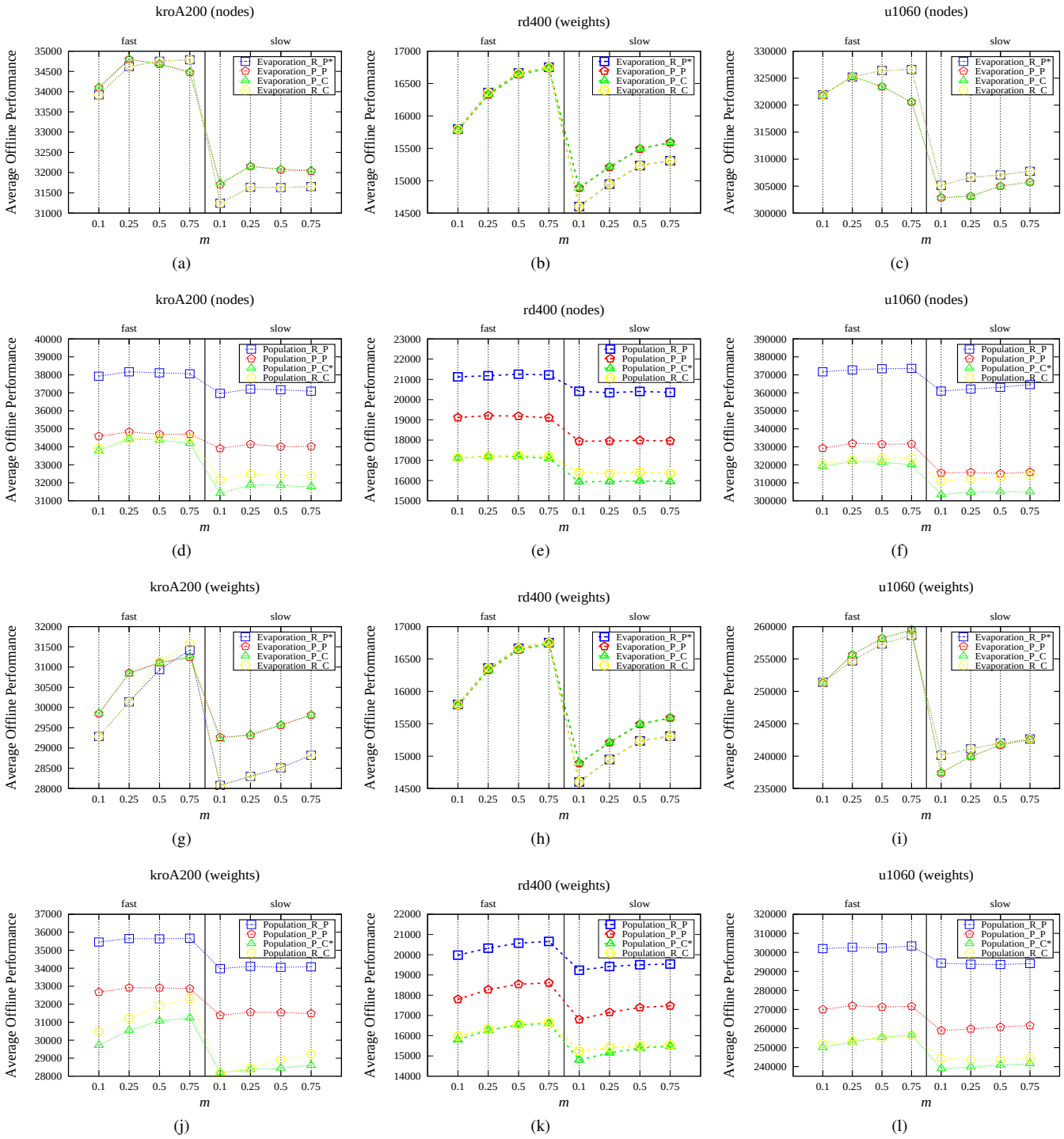


Fig. 1: Averaged (over 30 runs) offline performance for evaporation-based (a)(b)(c)(g)(h)(i) and population-based (d)(e)(f)(j)(k)(l) frameworks with alternative decision rules and pheromone update policies for different DTSPs. *These combinations are the default ones.

C. Effect of Dynamic Strategies

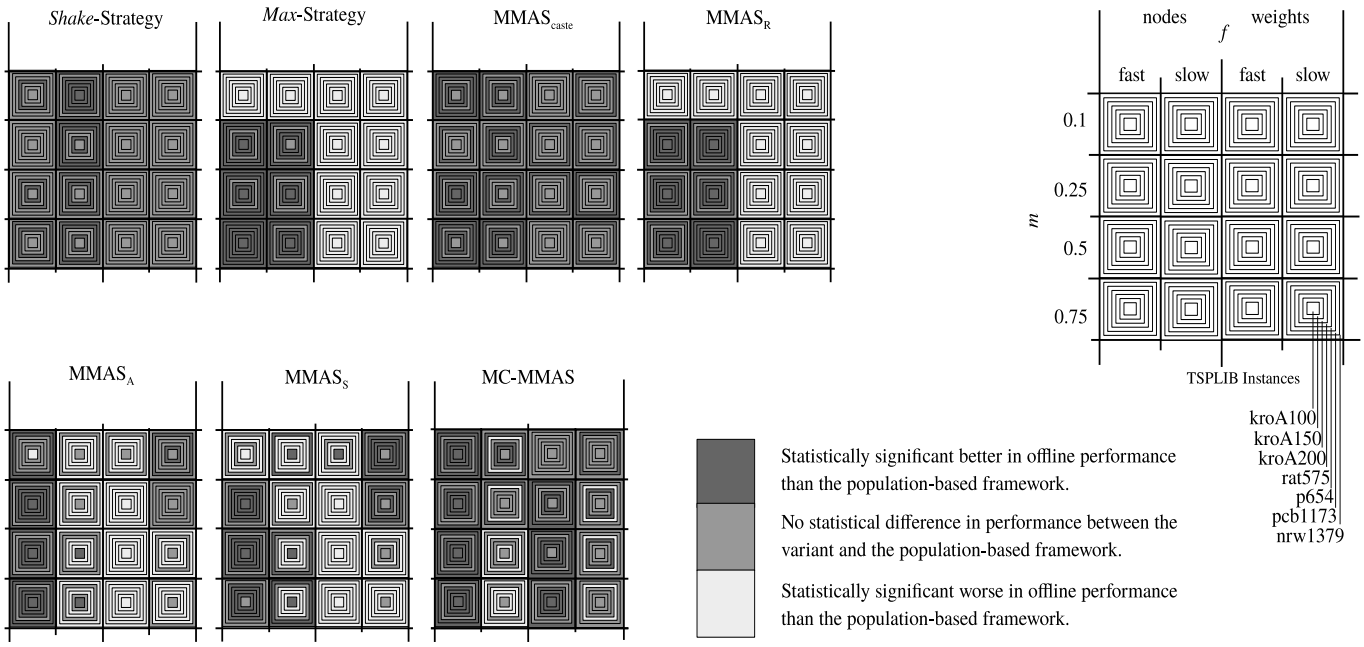


Fig. 2: Each square represents the comparisons of the statistical tests of the aforementioned ACO variant against the evaporation-based framework. Each square is subdivided into sixteen smaller squares that represent the dynamic settings of the DTSP. The squares are grouped by the type of change. Each smaller square contains a stack of increasingly larger boxes that represents a set of increasingly larger problem instances.

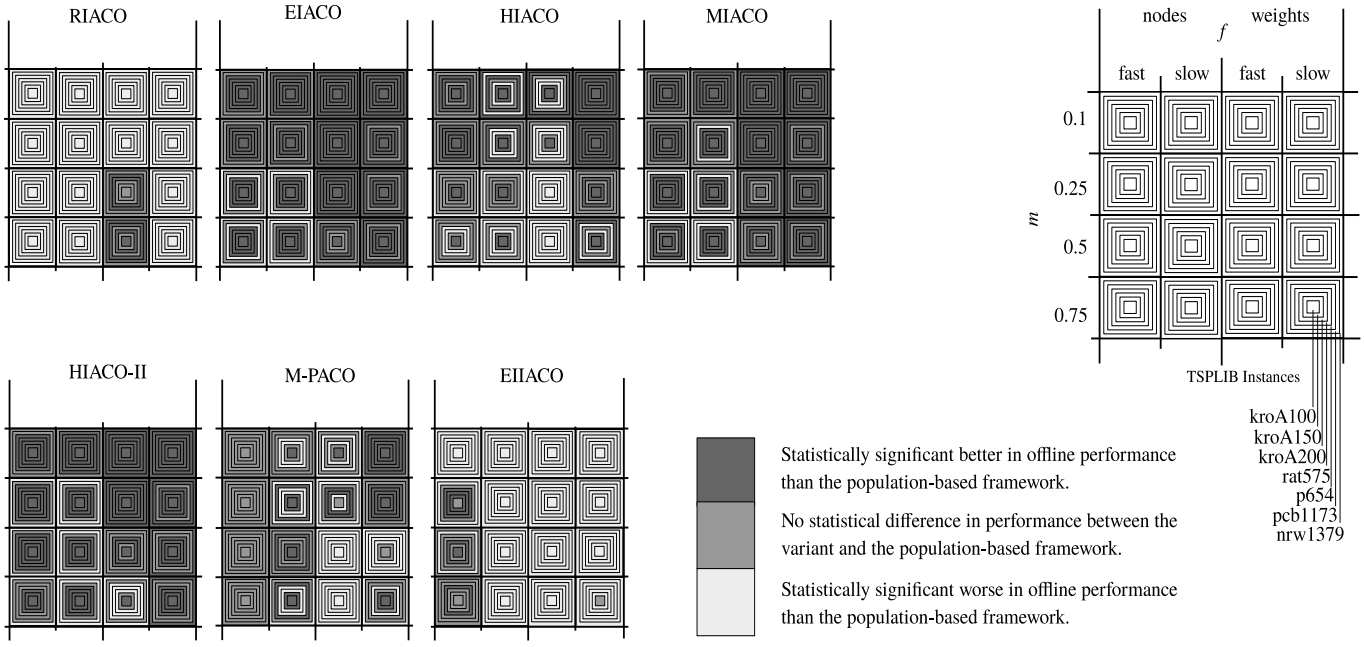


Fig. 3: Each square represents the comparisons of the statistical tests of the aforementioned ACO variant against the population-based framework. Each square is subdivided into sixteen smaller squares that represent the dynamic settings of the DTSP. The squares are grouped by the type of change. Each smaller square contains a stack of increasingly larger boxes that represents a set of increasingly larger problem instances.

D. Effect of Utilizing Change-Related Information

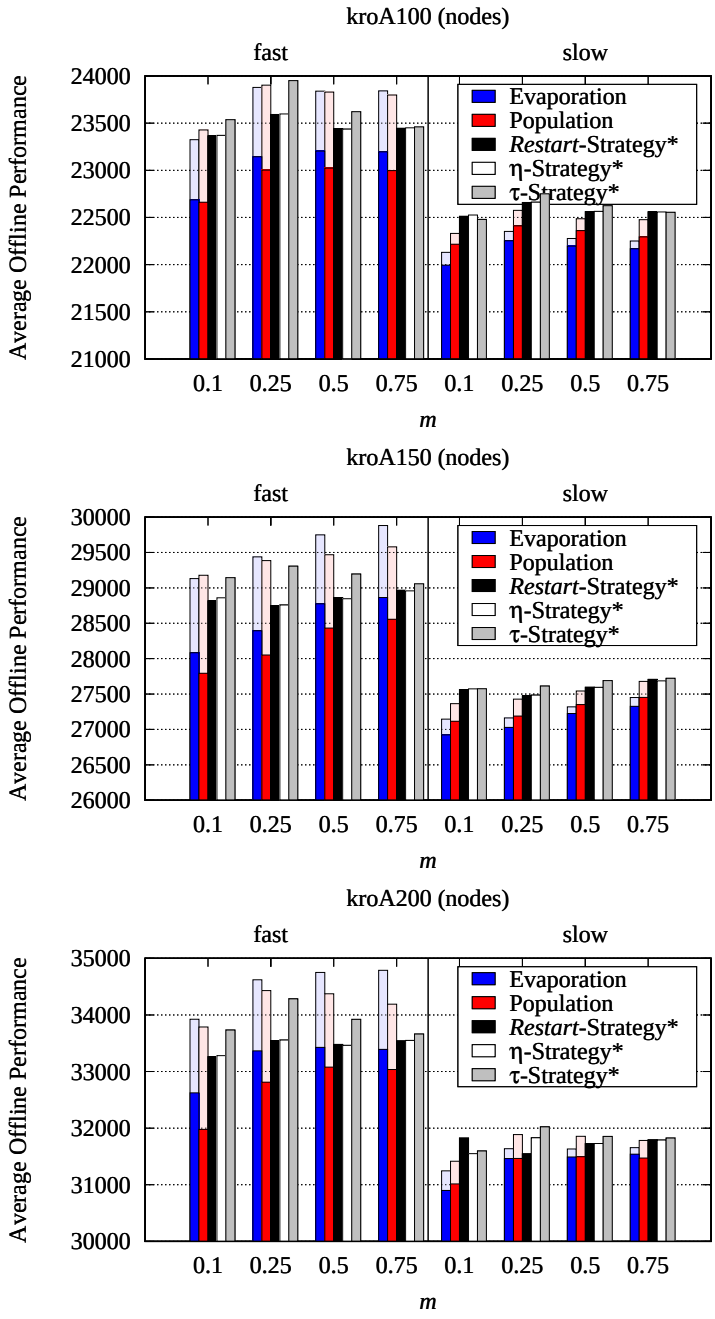


Fig. 4: $\bar{P}_{offline}$ (averaged over 30 runs) results of evaporation-based and population-based frameworks, and three evaporation-based variants when utilizing change-related information for DTSPs with node changes. Each bar is divided into two parts that represent the results when utilizing change-related information (darker) or not (lighter). *These strategies have been designed to utilize change-related information and, thus, the values when information is not utilized do not exist.

E. Comparisons with Evolutionary Algorithms

TABLE V: Experimental results regarding the $\bar{P}_{offline}$ (averaged over 30 runs) of ACO algorithms with state-of-the-art evolutionary algorithms for DTSPs with weight changes with $f = n \cdot 100$ and m randomly chosen from a uniform distribution in $(0.0, 0.5]$.

TSPLIB Instance	P-ACO	$\mathcal{MM}$AS	EIGA	GPX	EIACO	MC-$\mathcal{MM}$AS
berlin52	7261	7195	7414	7392	**7177**	7191
eil101	572	568	581	578	569	**567**
kroB200	28481	28261	29161	29026	**28231**	28345
lin318	40154	39957	41543	41054	40456	**39932**
pr439	105376	104591	105904	106193	104633	**103918**
p654	49138	49415	48178	**47921**	49127	49533
rat783	**8434**	8521	8515	8509	8436	8444
pr1002	270701	274370	281952	279321	**268532**	274301
u1432	158822	160881	163427	161203	**157503**	161244
DTSP with Node Changes						
berlin52	8080	8046	8749	8700	**8004**	8034
eil101	560	558	599	596	**555**	557
kroB200	31465	31245	33890	33632	**31095**	31278
lin318	47854	**47496**	49434	49345	47593	47499
pr439	159001	**156316**	167630	167557	158538	157975
p654	66474	**66075**	69902	69834	66712	66338
rat783	8837	8634	9234	9237	**8609**	8755
pr1002	**308512**	309991	322655	320925	310153	309137
u1432	**157658**	157875	178913	176435	158894	157907

Bold values indicate statistical significance
Underline values indicate no statistical difference with the bold value