Ant Colony Optimization in Stationary and Dynamic Environments

by

Michalis Mavrovouniotis

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Department of Computer Science

University of Leicester

2013

Declaration of Authorship

The content of this submission was undertaken in the Department of Computer Science, University of Leicester, and supervised by Prof. Shengxiang Yang during the period of registration. I hereby declare that the materials of this submission have not previously been published for a degree or diploma at any other university or institute. All the materials submitted for assessment are from my own research, except the reference work in any format by other authors, which are properly acknowledged in the content.

Part of the research work presented in this submission has been published or has been submitted for publication in the following papers:

1. M. Mavrovouniotis, S. Yang. Ant colony optimization with immigrants schemes in dynamic environments, "*Proceedings of the 11th International Conference on Parallel Problem Solving from Nature* (PPSN XI)", LNCS 6238, Part II, pp. 371–380, Springer-Verlag, 2010.
2. M. Mavrovouniotis, S. Yang. Ant colony optimization with direct communication for the travelling salesman problem, "*Proceedings of the 2010 Workshop on Computational Intelligence* (UKCI2010)", pp. 1–6, IEEE Press, 2010.
3. M. Mavrovouniotis, S. Yang. Memory-based immigrants for ant colony optimization in changing environments, "*EvoApplications 2011: Applications of Evolutionary Computation*", LNCS 6624, Part I, pp. 324–333, Springer-Verlag, 2011.
4. M. Mavrovouniotis, S. Yang. A memetic ant colony optimization for the dynamic travelling salesman problem, "*Soft Computing - A Fusion of Foundations, Methodologies and Applications*", vol. 15, no. 7, pp. 1405–1425, Springer-Verlag, 2011.
5. M. Mavrovouniotis, S. Yang. An ant system with direct communication for the capacitated vehicle routing problem, "*Proceedings of the 2011 Workshop on Computational Intelligence* (UKCI2011)", pp. 14–19, 2011.
6. M. Mavrovouniotis, S. Yang. An immigrants scheme based on environmental information for ant colony optimization for the dynamic travelling salesman

problem, "*Proceedings of the 10th International Conference on Artificial Evolution* (EA-2011)", LNCS 7401, pp. 1–12, Springer-Verlag, 2012.

7. M. Mavrovouniotis, S. Yang. Ant colony optimization with immigrants schemes for the dynamic vehicle routing problem, "*EvoApplications 2012: Applications of Evolutionary Computation*", LNCS 7248, pp. 519–528, Springer-Verlag, 2012.

8. M. Mavrovouniotis, S. Yang. Ant colony optimization with memory-based immigrants for the dynamic vehicle routing problem, "*Proceedings of the 2012 IEEE Congress on Evolutionary Computation*", pp. 2645–2652, IEEE Press, 2012.

9. M. Mavrovouniotis, S. Yang, X. Yao. A benchmark generator for dynamic permutation-encoded problems. "*Proceedings of the 12th International Conference on Parallel Problem Solving from Nature* (PPSN XII)", LNCS 7492, Part II, pp. 508–517, Springer-Verlag, 2012.

10. M. Mavrovouniotis, S. Yang. Ant colony optimization algorithms with immigrants schemes for the dynamic travelling salesman problem, S. Yang and X. Yao (eds.), "*Evolutionary Computation for Dynamic Optimization Problems*", Chapter 13, pp. 331–357, Springer-Verlag, 2013.

11. M. Mavrovouniotis, S. Yang. Dynamic vehicle routing: A memetic ant colony optimization approach. A.S. Uyar, E. Ozcan and N. Urquhart (eds.), "*Automated Scheduling*", Chapter 9, Springer-Verlag, 2013.

12. M. Mavrovouniotis, S. Yang. Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors, "*Applied Soft Computing*", Elsevier, 2013.

Ant Colony Optimization in Stationary and Dynamic Environments

Michalis Mavrovouniotis

Abstract

The ant colony optimization (ACO) metaheuristic is inspired by the foraging behaviour of real ant colonies. Similarly with other metaheuristics, ACO suffers from stagnation behaviour, where all ants construct the same solution from early stages. In result, the solution quality may be degraded because the population may get trapped on local optima. In this thesis, we propose a novel approach, called direct communication (DC) scheme, that helps ACO algorithms to escape from a local optimum if they get trapped. The experimental results on two routing problems showed that the DC scheme is effective.

Usually, researchers are focused on problems in which they have static environment. In the last decade, there is a growing interest to apply nature-inspired metaheuristics in optimization problems with dynamic environments. Usually, dynamic optimization problems (DOPs) are addressed using evolutionary algorithms. In this thesis, we apply several novel ACO algorithms in two routing DOPs. The proposed ACO algorithms are integrated with immigrants schemes in which immigrant ants are generated, either randomly or with the use of knowledge from previous environment(s), and replace other ants in the current population. The experimental results showed that each proposed algorithm performs better in different dynamic cases, and that they have better performance than other peer ACO algorithms in general.

The existing benchmark generators for DOPs are developed for binary-encoded combinatorial problems. Since routing problems are usually permutation-encoded combinatorial problems, the dynamic environments used in the experiments are generated using a novel benchmark generator that converts a static problem instance to a dynamic one. The specific dynamic benchmark generator changes the fitness landscape of the problem, which causes the optimum to change in every environmental change. Furthermore in this thesis, another benchmark generator is proposed which moves the population to another location in the fitness landscape, instead of modifying it. In this way, the optimum is known and one can see how close to the optimum an algorithm performs during the environmental changes.

Acknowledgements

First, I would like to thank my supervisor Prof. Shengxiang Yang who gave me the appropriate direction to complete this thesis. In fact, he was the person that promoted me in this research area, since he supervised me in my undergraduate third year project.

I am also grateful to my second supervisor Prof. Thomas Erlebach for all his support, encouragement, and advice, especially on the last two years of my studies. I would also like to thank Dr. Fer-Jan De Vries for assessing my work at the end of each year and Prof. Rajeev Raman for his advice on the professional side of my course. Furthermore, I would like to thank Prof. Xin Yao for his suggestions regarding some future work and direction for my work.

Special thanks to the University of Leicester and the Department of Computer Science for the financial support of my studies and my attendance in several conferences around Europe.

I would also like to thank all the members and colleagues in the Department of Computer Science, and all my friends in Leicester that formed a friendly environment for me all these years of my studies.

Also, I appreciate the support from my family, my parents and my brother. Without their help all the things I have achieved would have been impossible. This thesis is dedicated to them, but especially is dedicated to my loving grandmother who passed away during my studies.

Finally, I would like to thank the ants! Without these small insects from nature, “moving” on the graphs of the different optimization problems, this thesis would not exist. Similarly with the ants that find the shortest path via pheromone trails, I hope that the pheromone trails of the earth will be generated as quickly as possible to find all the persons mentioned above again.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ACO	**A**nt **C**olony **O**ptimization
AS	**A**nt **S**ystem
EAS	**E**litist **A**nt **S**ystem
ACS	**A**nt **C**olony **S**ystem
AS$_{rank}$	**R**ank-based **A**nt **S**ystem
$\mathcal{MM}$**AS**	$\mathcal{MAX} - \mathcal{MIN}$ **A**nt **S**ystem
BWAS	**B**est-**W**orst **A**nt **S**ystem
P-ACO	**P**opulation-based **A**nt **C**olony **O**ptimization
EA	**E**volutionary **A**lgorithm
GA	**G**enetic **A**lgorithm
SA	**S**imulated **A**nnealing
TS	**T**abu **S**earch
EDA	**E**stimation of **D**istribution **A**lgorithm
BCO	**B**ee **C**olony **O**ptimization
LS	**L**ocal **S**earch
DC	**D**irect **C**ommunication
RIACO	**R**andom **I**mmigrants **ACO**
EIACO	**E**litist-based **I**mmigrants **ACO**
MIACO	**M**emory-based **I**mmigrants **ACO**
SOP	**S**tationary **O**ptimization **P**roblem
DOP	**D**ynamic **O**ptimization **P**roblem
TSP	**T**ravelling **S**alesperson **P**roblem

VRP	**V**ehicle **R**outing **P**roblem
COP	**C**ombinatorial **O**ptimization **P**roblem
EDO	**E**volutionary **D**ynamic **O**ptimization

Chapter 1

Introduction

The word *optimization* refers to the process of making something as good or effective as possible. In computer science and mathematics, optimization was one of the oldest scientific issues, whereas optimization problems have been a major issue to be solved, either empirically or mathematically. Many optimization problems of practical or theoretical importance are of a combinatorial nature, which involve finding the values of discrete variables such that a certain condition is satisfied.

Probably the most widely known combinatorial optimization problem (COP) is the travelling salesperson problem (TSP), in which the shortest cyclic tour among a set of cities has to be found. COPs are intriguing because they are easy to state but often very difficult to solve. For instance, there is no existing algorithm to find the optimal solution for the TSP within polynomial time. This is the case in most COPs, and led to the development of the $\mathcal{NP}$-completeness complexity theory [105].

COPs appear in many real-world applications and there is a need for algorithms to solve them efficiently. The algorithms to solve COPs are classified into the following categories, *exact* or *approximation*. Exact algorithms guarantee to provide the global optimum solution, but not necessarily efficiently, whereas approximation (otherwise known as heuristics) methods, often provide a close to the global optimum solution efficiently.

However, most real-world applications change over time which makes COPs even more difficult to solve. Such problems are called dynamic optimization problems (DOPs). In DOPs, an environment, including the objective function, the decision variables, the problem instance, the constraints, and so on, may vary over time, which may cause the optimum to move. In the contrast to COPs with static environments, for which the goal is to find the optimum solution efficiently, for COPs with dynamic environments, the goal is to track the moving optimum solution through the changing environment efficiently.

In the last decade, nature-inspired algorithms, which are approximation algorithms, have been widely used to address different DOPs. It is believed that such algorithms are able to adapt to changing environments, since they are inspired from nature which is a continuous adaptation process [17, 137]. Popular examples of such algorithms applied in COPs with dynamic environments are: evolutionary algorithms (EAs) [89, 129] which are inspired from natural evolution of survival of the fittest and ant colony optimization (ACO) algorithms [67] which are inspired from the foraging behavior of real ant colonies. EAs have been mainly established to address COPs with binary-encoded representation, whereas ACO to address COPs with permutation-encoded representation.

1.1 Ant Colony Optimization (ACO)

Real ant colonies communicate via *pheromone*, which is a chemical substance produced by ants. The more pheromone ants smell in an area, the higher the probability that ants follow that area. The ACO framework is based on this simple idea.

More precisely, ACO algorithms consist of a population of ants, where each one represents a solution of the problem to be solved. Starting from an empty solution each ant builds up a feasible solution. After building a solution, each ant updates its pheromone trails to mark its solution. The pheromone value depends on the quality of the solution, i.e., the higher the solution quality the more the pheromone value. In the next iteration, ants will consider the pheromone trails generated from the previous iteration.

1.1.1 Challenges for ACO in Static Environments

The research of ACO in static optimization problems (SOPs) grew up rapidly over the years, addressing a lot of applications and developing several ACO variations and extensions.

However, ACO algorithms suffer from the stagnation behaviour, where all ants follow the same path from the initial stages of the algorithm, when applied to different COPs. This is because a high intensity of pheromone is generated to a single trail quickly, that marks a potential solution, and attracts the ants to that area. Therefore, traditional ACO algorithms are more likely to get trapped in a local optimum solution, which may degrade the solution quality and the overall performance of the algorithm.

In general, it is difficult to achieve a balance between exploration and exploitation for ACO algorithms. Exploration is the ability of an algorithm to discover solutions not previously selected, whereas exploitation is the ability of an algorithm to search for solutions in areas where good solutions have previously been found. However, a proper trade-off between exploration and exploitation is important in order to achieve a robust searching behaviour for ACO algorithms, or any other metaheuristic in general.

1.1.2 Challenges for ACO in Dynamic Environments

Similarly, stagnation behaviour is a major problem for ACO in DOPs. In case the algorithm converges quickly, it loses its exploration, and, thus, its adaptation capabilities. This is because when the environment changes, the pheromone trails of the previous environment will not match the new environment, especially in the case where pheromone is located to a single trail.

However, ACO has an adaptation mechanism, called *pheromone evaporation*, in which a constant amount of pheromone is deducted from all trails. This may help eliminate pheromone trails that are not useful and bias the ants to non-promising areas. A typical way to address DOPs is to re-initialize the pheromone trails equally after a dynamic change, which acts as a restart of the algorithm. Therefore, every change is considered as the arrival of a new optimization problem that has to be solved from scratch. However, such strategy requires substantial computational efforts and the detection of change, which sometimes is difficult to detect. A more efficient way to address the stagnation behaviour and adapt well to DOPs is to transfer knowledge from previous environments. However, the algorithm needs to be robust enough to accept the knowledge transferred, which can be achieved by the maintenance of the solution's diversity. Of course, high levels of diversity may disturb the optimization process and lead the algorithm to randomization.

The research of ACO in DOPs is still in its infancy, with just a few applications and ACO variations. Most of the research in DOPs has been done with EAs, which is otherwise known as *evolutionary dynamic optimization* (EDO). EDO consists of many strategies integrated with EAs to address premature converge and enhance the adaptation capabilities of the algorithms. Similar strategies can be integrated with ACO algorithms.

1.2 Aims and Objectives

The main aim of this thesis is to develop effective approaches to avoid stagnation behaviour for COPs in static environments, and to contribute to the research of ACO in DOPs, with the development of strategies to enhance the adaptation capabilities of ACO. To achieve these aims the following objectives are established:

- Understand the search behaviour of ACO in routing problems (initially under static environments).
- Improve the performance of ACO in static environments to delay convergence and avoid stagnation behaviour.
- Develop a benchmark for permutation-encoded DOPs in order to test the algorithms in dynamic environments.
- Develop pheromone strategies (mainly inspired from EAs) to enhance the adaptation capabilities of ACO algorithms.
- Establish an ACO framework for DOPs.
- Perform systematic experimental studies and relevant analysis to justify proposed ACO variants in both SOPs and DOPs.

1.3 Scientific Contributions

In this thesis we propose and present the following approaches to achieve the aforementioned objectives:

- For SOPs

  - A scheme where ants, apart from communicating indirectly via pheromone trails, communicate directly by exchanging information that improves the performance of traditional ACO in SOPs.

- For DOPs
  - An ACO framework to address DOPs.
  - The integration of immigrants schemes with ACO for DOPs.
  - Development of a benchmark generator that models a real-world scenario.
  - Development of a benchmark generator where the optimum is known in every environmental change.

1.4 Structure of the Thesis

The rest of this thesis is organised as follows.

In Chapter 2, the biological roots of ACO algorithms are presented, including the early biological experiment of ants finding the shortest path and the first probabilistic model inspired from ant's behaviour. Moreover, a simple ACO algorithm is described and it can be clearly observed what are the differences in the behaviour of real ant colonies in comparison with the artificial ant colonies.

In Chapter 3, combinatorial optimization is described including the examples of problems with their applications. Furthermore, an explanation why these problems are hard to solve is given with respect to some methods. Finally, the term metaheuristic is explained with several examples, including their characteristics.

In Chapter 4, the ACO metaheuristic is described, including its historical contributions. The first ACO algorithm with its several variations and extensions is described for the fundamental TSP. Furthermore, a survey of the main contributions regarding the ACO applications for SOPs is given, whereas the application for the TSP and the vehicle routing problem (VRP) are explained in more detail because they are the problems used in the experiments in Chapter 5.

In Chapter 5, the proposed scheme based on the direct communication (DC) between ants is described for static TSPs and VRPs. Furthermore, experiments are presented, including the optimization of some parameter settings, comparisons between the different variations of traditional ACO algorithms, and comparisons between conventional ACO and ACO with the DC scheme for the two optimization problems.

In Chapter 6, the concept of DOPs is explained, including the methods and strategies used, how to detect an environmental change and the performance measurements used. Furthermore, an explanation why ACO algorithms are suitable for DOPs is given, followed by a survey of the main contributions regarding the applications of ACO in DOPs. As in Chapter 4, the applications for TSP and VRP, in this case the dynamic versions, are explained in more detail because they are the problems that will be used in the experiments in Chapter 8.

In Chapter 7, the need of benchmark generators is explained. The properties of a good benchmark generator are explained, including a description of one of the most common benchmark generators used in binary-encoded DOPs. Furthermore, the proposed benchmark generators for dynamic TSPs (DTSPs) and dynamic VRPs (DVRPs) are described, in which for the first one the optimum is known during the changes, whereas for the second one, the optimum is unknown in every environmental change.

In Chapter 8, the proposed algorithms, where immigrants schemes are integrated to an ACO framework, designed especially for DOPs, are described for the DTSP and

DVRP. In the experiments, the algorithms are tested in different dynamic cases and environmental types using a benchmark generator from Chapter 7. Furthermore, we show the impact of the important parameters of the algorithms. Finally, the proposed algorithms have been compared with other peer ACO algorithms in both DTSP and DVRP and showed good performance.

Finally, Chapter 9 concludes the thesis with the technical contributions and the outcome of the experimental results of this thesis, and gives some discussion regarding the future work.

Chapter 2

From Natural to Artificial Ant Colonies

2.1 Real Ant Colonies Behaviour

The main characteristic of the behaviour of many ant species, e.g., Argentine linepithema humile, is *foraging* [17, 128]. This behaviour is based on the communication achieved by ants via a chemical substance produced by themselves, called *pheromone*. While ants walk to search for food sources from their nest and vice versa, they lay pheromones on the ground to mark their path, forming in this way different pheromone trails. Ants can smell pheromones, including the pheromone of other ants, when walking and they are usually attracted to the areas with strong pheromone concentrations. In this way, the ants are able to find their way back to the nest or to the food sources.

This “reading-” and “writing-” pheromone behaviour of ants has inspired researchers to design and run foraging experiments [53, 110], to observe whether the ants in the colony as a whole are able to discover the shortest path when few alternative paths

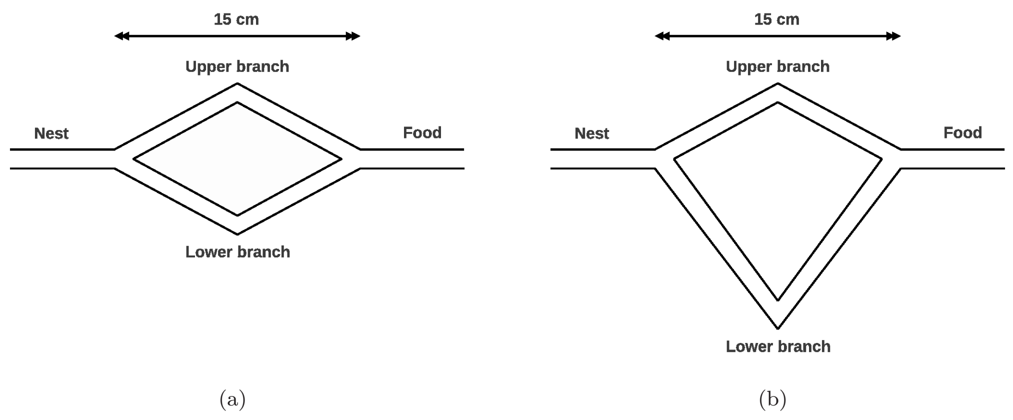


Figure 2.1: Double bridge experiments setup: 2.1(a) the two branches have equal length and 2.1(b) the two branches have unequal length.

exist from colony's nest to a food source. The experiments use a colony of Argentine ants and a double bridge with two branches where both connect the colony's nest with a food source; see Figure 2.1.

2.1.1 Behaviour in Static Environments

2.1.1.1 Double Bridge Experiment with Branches of Equal Length

Deneubourg et al. [53] run an experiment where the bridge has two branches of equal length that connect the colony's nest with a food source as shown in Figure 2.1(a). Since the branches have the same length, the two paths generated have the same length as well.

Ants are released to move between the nest and the food source. The percentage of ants that chose the upper or the lower branch is recorded over time. The outcome from the initial phase is that at the beginning the ants are choosing either the upper or the lower branch, randomly. This is because initially there is no pheromone on any of the two branches to attract the ants to chose the shortest one. Therefore, the ants select with an equal probability any of the branches as shown in Figure 2.2(a).

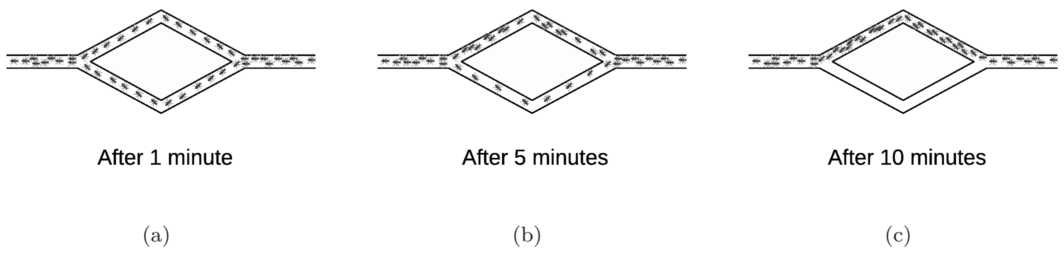


FIGURE 2.2: Double bridge experiment in a static environment with equal branches.

After a few minutes from the initial phase more ants follow the same path. In this case, is the path generated with the upper branch, as shown in Figure 2.2(b). This is because a few more ants have selected the upper branch on the initial phase where ants' choices were random. Therefore, a larger number of ants in the upper branch means higher intensity of pheromones since the ants deposit pheromone while they are walking. After a few more minutes the ants will have a preference regarding which branch to select, since the upper one will contain more pheromone than the lower one, as shown in Figure 2.2(c).

2.1.1.2 Double Bridge Experiment with Branches of Unequal Length

Gross et al. [110] run an experiment similar with the above one, where the bridge has two branches of unequal length. This time the lower branch is twice the length of the upper branch as shown in Figure 2.1(b).

Similarly to the first experiment with the equal branches there is no pheromone on the two branches. Therefore, on the initial phase the ants are randomly choosing either the upper or the lower branch since they appear identical to them, as shown in Figure 2.3(a). However, the outcome after a few minutes from the initial phase was that the ants chose the shorter branch, even if a few more ants randomly chose the longer branch in the initial phase as in the first experiment, as shown in Figure 2.3(b).

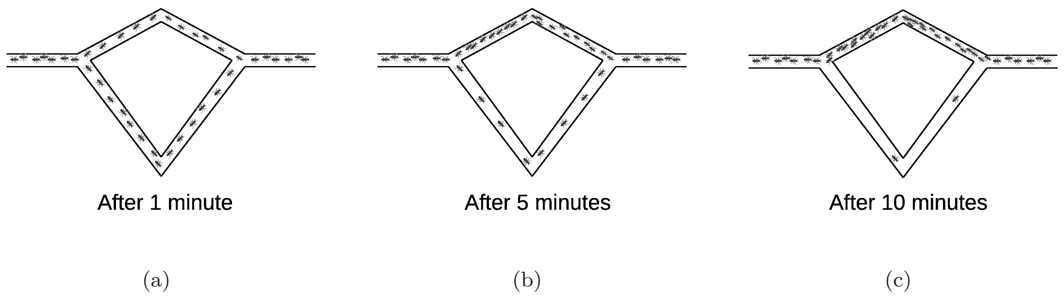


FIGURE 2.3: Double bridge experiment in a static environment with unequal branches.

The reason why ants chose the upper branch is that the ants of the shorter branch are the first to reach the food and return to the nest. Since ants deposit pheromone while they are walking, each ant will deposit twice pheromone to the shorter branch, i.e., one time walking from nest to the food source and another time vice versa. Therefore, when they are about to make a decision between the two branches again, the shorter one will contain higher intensity of pheromone from the longer one because the ants from the longer branch will still be on their way back to the nest. As a result, the pheromone on the shorter branch is increased faster and attracts the ants after a few minutes of the initial phase. Moreover, it was observed that not all the ants converged to the path of the shorter branch, since a very small percentage was still choosing the longer path, as shown in Figure 2.3(c).

2.1.2 Behaviour in Dynamic Environments

2.1.2.1 Experiment with Equal Pheromone on the Paths

Considering the double bridge experiments it is confirmed that a colony of ants is able to find the shortest path without using any visual cues, but via their pheromone trails [128]. From the observations in these experiments [53, 110] we can also claim

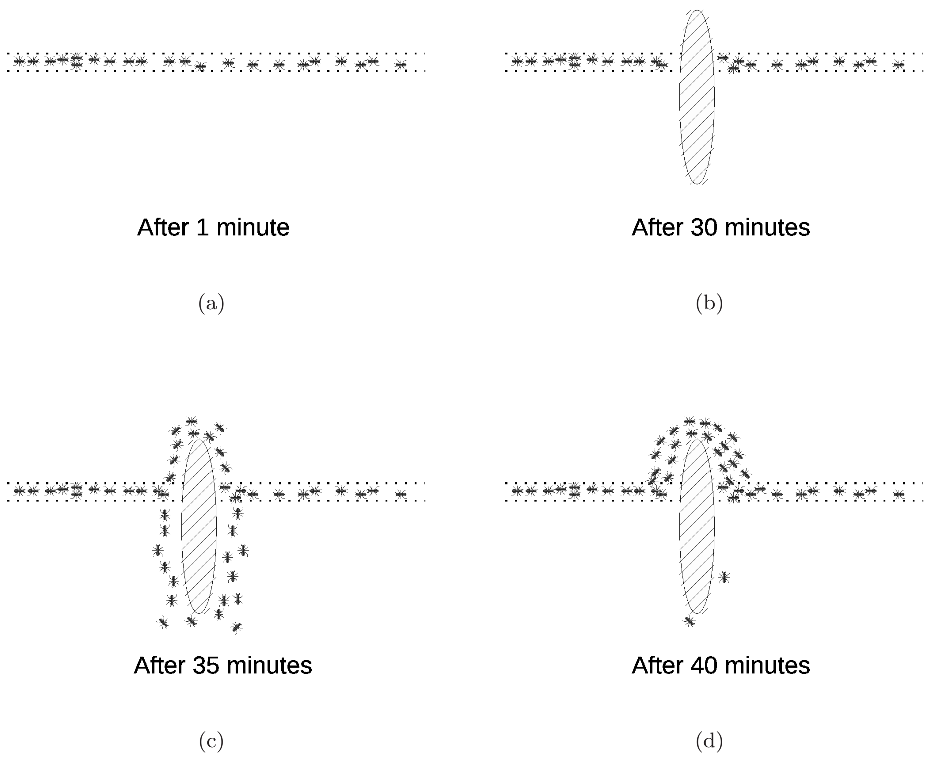


FIGURE 2.4: Double bridge experiment in a dynamic environment with equal pheromone trails on the paths constructed.

that they can adapt to dynamic environments, e.g., to find an alternative path when the current shortest path found by the ants becomes infeasible.

In Figure 2.4(a) the ants move and converge to the path that connects their nest to a food source. In result, high intensity of pheromone will be generated to the specific path since the ants deposit pheromone to their trails when they walk. After a few minutes an obstacle appears in the middle of the path and the ants are not able to move on the specific path, as shown in Figure 2.4(b). As on the double bridge experiments it is expected that approximately half of the ants on the front of the obstacle will turn right from the obstacle and the remaining will turn left. The same is expected from the ants on the other side of the obstacle, as shown in Figure 2.4(c). Since the path formed on the left side around the obstacle is shorter,

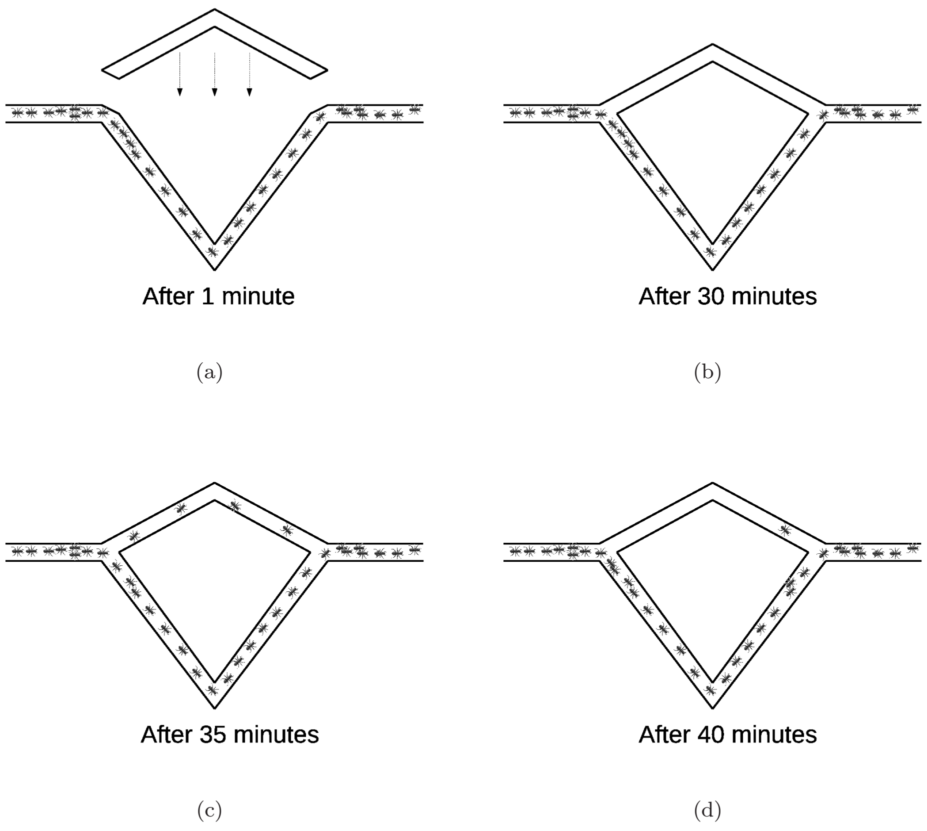


FIGURE 2.5: Double bridge experiment in a dynamic environment with unequal pheromone trails on the paths constructed.

the pheromone trails will be generated faster than the ones on the right side around the obstacle considering the observation from the "double bridge experiments with branches of unequal length". As a result, almost all the ants will follow the shortest path, as shown in Figure 2.4(d).

2.1.2.2 Experiment with Unequal Pheromone on the Paths

In fact, Gross et al. [110] performed an additional experiment to confirm whether ants can adapt to dynamic changes. Similarly with the double bridge experiment with unequal branches design, in this experiment only the long branch was offered to the colony initially. Since only one path is available between the nest and the food source the ants will converge to that, as shown in Figure 2.5(a). After 30 minutes the short branch was added, as shown in Figure 2.5(b). Considering the obstacle example in Figure 2.4 the ants are expected to select the path of the short branch, but that was not the case. The path of the short branch was only selected by few ants and not very frequently, whereas the path of the long branch was selected by almost all ants, as shown in Figures 2.5(c) and 2.5(d).

The difference of this experiment with the experiment of the obstacle, in Figure 2.4, is that pheromone exists in one of the two paths generated after the change of the environment. This can be explained by the high intensity of pheromones generated to the long branch which still attracts the ants even if a shorter path appears, and the slow evaporation of pheromone. In the experiment it was observed that the lifetime of the pheromone is comparable to the duration of an experimental trial.

Therefore, if the pheromone evaporation was faster, the pheromone trails in the path of the long branch will be eliminated quickly and the probability to discover the path of the short branch will be increased.

2.2 A Probabilistic Model for Artificial Ants

From the observations derived in the double bridge experiments, a simple probabilistic model has been proposed to describe the dynamics of real ants [53]. In the model, the following two assumptions are taken:

- The pheromone on the two branches is proportional to the number of ants that have used them to cross the bridge in the past until that moment.
- The pheromone does not evaporate, since the experimental time is not enough to cause the pheromone to evaporate.

More precisely, let μ be the size of the ant colony and U_μ and L_μ be the number of ants that have used the upper branch and lower branch, respectively, where $\mu = U_\mu + L_\mu$. The probability $p_U(\mu)$ of the $(\mu+1)$-th ant to choose the upper branch is:

$$p_U(\mu) = \frac{(U_\mu + t_d)^\alpha}{(U_\mu + t_d)^\alpha + (L_\mu + t_d)^\alpha}, \tag{2.1}$$

where t_d represents time delay of the ant to traverse the upper branch and $\alpha = 2$ is a parameter to fit the model with the double bridge experiment, since ants deposit pheromone twice, i.e., from nest to food source and vice versa. The corresponding probability $p_U(\mu)$ of for the lower branch is:

$$p_L(\mu) = 1 - p_U(\mu). \tag{2.2}$$

The number of ants choosing the upper branch is defined as:

$$U_{\mu+1} = \begin{cases} U_\mu + 1, & \text{if } R \le p_U(\mu); \\ U_\mu, & \text{otherwise,} \end{cases} \tag{2.3}$$

and the number of ants choosing the lower branch is defined as:

$$L_{\mu+1} = \begin{cases} L_\mu + 1, & \text{if } R > p_L(\mu); \\ L_\mu, & \text{otherwise,} \end{cases} \tag{2.4}$$

where R is a random number uniformly distributed in the interval $[0, 1]$.

Considering the equations of the model above, further simulations using the Monte Carlo method [161], with 1000 simulations and the environment used in the double bridge experiments, confirmed that the model matches the observations discovered in real ant colonies (when $\alpha \approx 2$ and $t_d \approx 20$) [110].

2.3 Artificial Ant Colonies Behaviour

2.3.1 Real Ants vs Artificial Ants

The behaviour of real ant colonies in the double bridge experiments shows properties, that can be used to design *artificial* ants and use them to solve path optimization problems, listed as follows:

- *Optimization* because they are able to find the shortest path.
- *Exploitation* because they prefer the path rich in pheromone and converge to it.
- *Exploration* because even when they converge to a path a few ants may still choose another path due to their stochastic decisions.
- *Adaptation* because of pheromone evaporation which may eliminate pheromone trails that bias ants not to follow the shortest path (this may take a lot of time due to slow pheromone evaporation).
- *Memorization* because they mark their path with pheromone trails as they walk.
- *Iterative* because they walk from their nest to the food source several times.

Therefore, to design an artificial ant with the above properties for path optimization, a fully connected graph is considered with several nodes as in Figure 2.6. The idea is to let each ant “walk” on the arcs of the graph, from the nest (start node), visiting a neighbour node on each step until it finds the food source (end node). Each ant is able to “read” existing pheromone and “write” its own pheromone on the arcs when walking.

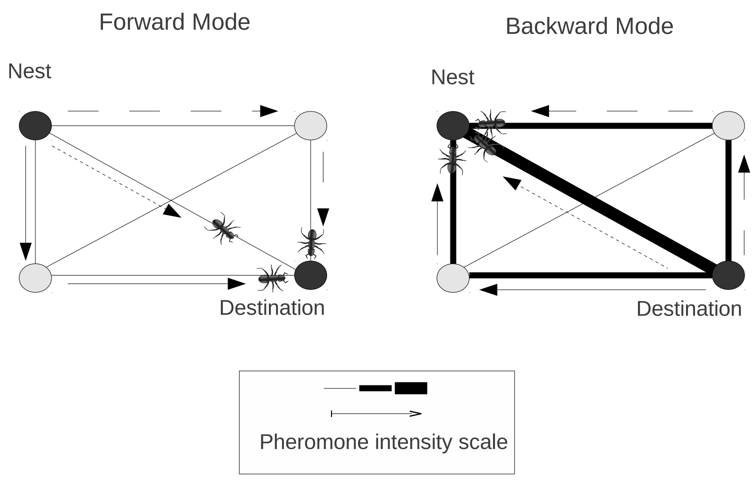


Figure 2.6: Example of how ants "walk" on graphs. On the forward mode they build solutions and on the backward mode they deposit pheromone. The intensity of pheromone varies, where the ant that constructed the shortest tour deposits more pheromone.

However, if the artificial ant's behaviour is the same as real ant's behaviour, it is less likely to find the shortest path on more complex graphs, where more than two paths are possible. This is because the ants may get trapped in infinite loops when they walk on the graph to generate a solution, i.e., walking on the same arc (or arcs) all the time. In result, high amounts of pheromone will be generated in the path, most probably not the shortest one, where the ants get stuck since they deposit pheromone as they walk. The problem is due to the implicit memory the ants have by marking their path with pheromone and they can revisit partial paths several times when they build a solution.

It is essential to extend the capabilities in artificial ants to address this problem, but retain the properties of real ants, in order to build feasible paths on graphs until they converge to the shortest one. Therefore, artificial ants use explicit memory in which they can store the nodes visited so far, representing a partial path. In this way, the problem of the infinite loops is solved because artificial ants will not be allowed to visit nodes stored in their memory.

Furthermore, while ants build their solution they do not deposit any pheromone, until they build a feasible path. Since the explicit memory of the ant will contain a valid solution it is then possible to evaluate it, e.g., the sum of the length of the the arcs, in order to deposit pheromone proportional to the solution's quality. Moreover, it enables the ants to retrace their solutions and deposit the appropriate pheromone on the arcs of their solutions. The extended capabilities of the artificial ants satisfy the following behaviour: the shorter the path constructed the more amount of pheromone to be deposited to the trail of that path.

2.3.2 Artificial Ants for the Shortest Path

Let $G = (V, E)$ be a graph where V is a set of nodes and E is a set of arcs. Each arc $(i, j) \in E$ between nodes i and j is associated with *artificial pheromone*, i.e., τ_{ij}, with $i, j \in V$. Initially, all arcs are assigned with an equal amount of pheromone. Each ant consists of two modes.

On the *forward* mode ants construct solutions probabilistically, starting from the start node. The probability of ant k to choose the next node j when its current node, i.e., the last node stored in its memory, is i, is calculated as follows:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha}{\sum_{l \in \mathcal{N}_i^k} \tau_{ij}^\alpha}, & \text{if } j \in \mathcal{N}_i^k; \\ 0, & \text{if } j \notin \mathcal{N}_i^k, \end{cases} \tag{2.5}$$

where $\mathcal{N}_i^k$ is the neighbourhood of unvisited nodes of ant k when its current node is i, τ_{ij} is the amount of existing pheromone in arc (i, j) and α is a constant parameter. Each ant repeats the node-by-node decision mechanism until it reaches the destination node, i.e., when $\mathcal{N}_i^k$ is empty. However, it may be the case that $\mathcal{N}_i^k$ is empty but the ant did not reach the destination node. This corresponds to a dead end in the graph, and, thus, the predecessor of node i is added into $\mathcal{N}_i^k$ to enable the ant to escape from it.

On the *backward* mode ants retrace their path to deposit pheromone from the destination node to the start node. In case the path contains a dead end then it is removed. An ant k updates the existing values of τ_{ij}, which correspond to the arcs of the path as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau^k, \tag{2.6}$$

where $\Delta\tau^k$ is the value where the existing τ_{ij} is increased. The value can be proportional to the quality of the solution or constant. Additionally, pheromone evaporation is performed where the intensity of all the existing pheromone trails is reduced as follows:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \forall(i,j) \in E, \tag{2.7}$$

where $\rho \in (0, 1]$ is a constant parameter that represents the evaporation factor. Evaporation is applied when all ants reach the destination node and complete their forward mode. After evaporation, the ants switch to their backward mode to deposit pheromone and reach the start node, where their memory is cleared. An *iteration* t is completed when all ants complete their backward mode. Another iteration $t + 1$ starts using the pheromone trails generated by the ants up to iteration t.

2.4 Summary

In this chapter we have presented the behaviour of real ant colonies and the first simple algorithm based on ants' foraging behaviour. From the double bridge experiments we have seen that real ant colonies can find the shortest path. Similarly, the experiments for artificial ant colonies based on a simple probabilistic model show, to some extent, the same behaviour.

In general, nature can teach us a lot on the way to solve problems in the real-world. For instance, it is important to have a colony size larger than one in order to solve problems. Another important aspect is the pheromone evaporation, which helps the ants to forget bad decisions made in previous iterations. Hence, this can be interpreted as a benefit for both static and dynamic graphs. On static graphs it may help to avoid get trapped in a local optimum and improve the solution quality, whereas on dynamic graphs it may help to eliminate pheromone trails that bias the ants not to adapt well to the newly generated graph.

Moreover, on static graphs the pheromone updates with a proportional amount of pheromone may work in favour of the detection to find the shortest path faster. Differently on dynamic graphs, where the cost of the arcs may change, the pheromone updates with a constant amount of pheromone seems a better choice because any bad solution on an old environment may be good for the new one. Hence, equal probabilities are given for the decision of the ants in the new environment.

Chapter 3

Combinatorial Optimization and Metaheuristics

3.1 Combinatorial Optimization

In general, an optimization problem can be defined as the problem of finding the best solution, called *global optimum*, from a set of feasible solutions. Formally an instance of an optimization problem can be defined as:

$$\Pi = (X, \Omega, f), \tag{3.1}$$

where Π is the optimization problem, X is a set of solutions, called the *search space*, Ω is a set of constraints and f is the objective function which assigns an objective value $f(x)$ to each solution $x \in X$. A feasible solution x is a set of optimization variable values $x = \{x_0, \dots, x_n\}$, that satisfies the constraints Ω. If Ω is empty, then the problem is called *unconstrained*, in which no constraints need to be satisfied. An optimization problem can be either a maximization or a minimization problem, such that $f(x^*) \geq f(x), \forall x \in X$ or $f(x^*) \leq f(x), \forall x \in X$, respectively, where x^*

is the global optimum solution. A COP can be defined as an optimization problem with a finite set of discrete optimization variables. Note that in this thesis, we only consider minimization COPs that are modelled using weighted graphs, i.e., routing problems.

3.1.1 Examples of Routing Problems

Routing problems are usually represented by a complete weighted graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ is a set of n nodes and $E = (i, j) : i \neq j$ is a set of arcs connecting the nodes. Each arc (i, j) is associated with a weight d_{ij} which is the distance between i and j. Typical examples are the TSP and its variants, e.g., the VRP.

Intuitively, the TSP is the problem in which a salesperson wants to find the shortest path through a given set of customers in different cities, starting from his home city, by visiting each city once before finally returning home. Differently, the VRP is the problem in which a number of vehicles with limited capacity are routed in order to satisfy the demand of all customers at a minimum cost (usually the total travel time), starting from and returning to the depot. For a formal description of these problems see Sections 4.6.1 and 4.6.2.

3.1.2 Applications

Considering that many problems have a finite number of alternative solutions to be addressed, they can be formulated as COPs. Many practical problems arise from physical networks, such as streets, railway stations, communication networks, etc. Such problems can be easily represented with graphs. An example of a COP with a physical network that is represented on a graph is the TSP.

The TSP is a classical optimization problem that has wide applications in routing and scheduling, such as the VRP which is closely related in the field of transportation, distribution of goods and logistics [38]. The arc routing counterpart of the VRP, i.e., the capacitated arc routing problem, has also many applications in the real-world including salt routing optimization [108, 122], urban waste collection [55, 190] and snow removal [34, 212].

3.2 Computational Complexity

COPs are usually easy to state, since they are characterized by a finite set of feasible solutions, but they are very difficult to solve by algorithms. The challenge arises from the exponential growth of the number of possible solutions as the problem size increases [142]. For example, in a TSP with n cities, the number of possible feasible solutions is $n!$. Hence, evaluating all the possible solutions, using exhaustive search, for a large problem instance to find the best one becomes impractical because of efficiency (running time).

The efficiency of an algorithm for a given problem Π is usually measured using the *worst-case complexity*. In other words, the worst-case complexity is the longest running time needed by the algorithm to find a solution for Π of any input size n. Usually, the worst-case complexity is formalized using the $\mathcal{O}(\cdot)$ notation. Let $g(n)$ and $h(n)$ be functions from the positive integers. Then, we say $g(n) = \mathcal{O}(h(n))$ if and only if $g(n) \leq \Phi \times h(n), \forall n \geq n_0$, where Φ and n_0 are two positive constants.

Moreover, the theory of $\mathcal{NP}$-completeness characterizes the difficulty of COPs [105], and distinguishes between two classes of problems: the class $\mathcal{P}$ in which a COP is solvable by a deterministic algorithm in polynomial time, i.e., the maximum amount of computation time needed to find the global optimum of any instance of size n of

Π is bounded by a polynomial in n, and the class of $\mathcal{NP}$ in which a COP is solvable by a non-deterministic algorithm in polynomial time.

A typical example of a $\mathcal{P}$ problem is the simple shortest path problem which can be solved by Dijkstra's algorithm in polynomial time [76]. For the great majority of COPs the computation time of the best algorithms known increases exponentially as the size of the problem instance increases, such as the TSP which is an $\mathcal{NP}$ problem. Therefore, no polynomial bound considering the worst-case complexity has been found so far. For example, the computation time of exhaustive search in the TSP is $\mathcal{O}(n! \times n)$, and, thus, the time to find the global optimum solution increases exponentially. A faster algorithm based on dynamic programming solves the TSP in $\mathcal{O}(n^2 2^n)$ [125].

Although it was proved that $\mathcal{P} \subseteq \mathcal{NP}$ [105], after decades of research efforts there is no evidence that prove that $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$. Since no polynomial-time algorithm has been found yet to solve any problems in $\mathcal{NP}$, most researchers tend to accept and assume that $\mathcal{P} \neq \mathcal{NP}$, but still it is an open research question. Often, the hardest problems classified in $\mathcal{NP}$, are called $\mathcal{NP}$-complete.

3.3 Methods for $\mathcal{NP}$-complete Problems

Solving $\mathcal{NP}$-complete COPs is a very challenging task due to their time complexity. Moreover, they may contain many local optima solutions which make it even more difficult for an algorithm to reach the global optimum solution; see Figure 3.1. Formally, a local optimum solution can be defined as a solution x such that $f(x) \neq f(x^*)$ and $f(x) \leq f(x'), \forall x' \in \mathcal{N}(x)$, where x^* is the global optimum which is defined as $f(x^*) \leq f(x'), \forall x' \in X$ and $\mathcal{N}(x)$ is the neighbourhood of a solution x which is

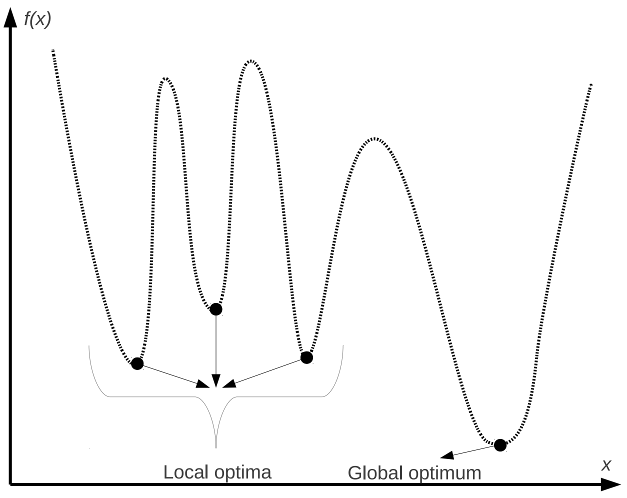


Figure 3.1: Illustration of local optima and a global optimum.

defined as a function $\mathcal{N} : X \mapsto 2^X$ which assigns to every $x \in X$ a set of neighbours $\mathcal{N}(x) \subseteq X$.

An algorithm may consider a very poor-quality local optimum solution as the global one and get stuck to it. There are two classes of solution methods to address COPs:

- *Exact* methods: guarantee to provide the global optimum solution, but often not efficiently; and
- *Heuristic* methods: do not guarantee to provide the global optimum solution, but often provide a solution near the global optimum efficiently.

Due to the $\mathcal{NP}$-completeness of most COPs, exact methods need, in the worst case scenario, exponential time to find the global optimum which makes them impractical. Such methods suffer from the growth of computation time as the size of COPs increases. However, with the increasing power of parallel computers the running time of some exact methods has been improved significantly over the years. For

Algorithm 1 Greedy Constructive Heuristic

```
1: $x \leftarrow$ empty solution
2: $x_0 \leftarrow$ InitialComponent
3: $x \leftarrow x \cup x_0$
4: while ($x$ is not a complete solution) do
5:     $x_c \leftarrow$ SelectGreedyComponent($x$)
6:     $x \leftarrow x \cup x_c$
7: end while
8: return $x$
```

example, in 2006 a TSP instance of 85,900 points was solved to optimality taking 136 years of CPU-Time [5].

On the other hand heuristic methods sacrifice optimality for the sake of efficiency. In other words, they are able to provide a near-optimum solution in polynomial time. For example, the Lin-Kernighan heuristic can provide near-optimum solutions for the TSP in a few seconds of computation time [126]. Therefore, heuristics are useful on difficult COPs, especially large problem instances, in which exact methods require extensive run time and huge computational efforts. Typically, heuristic methods are classified into two classes:

- *Constructive* Heuristics: that generate solutions from scratch by iteratively adding solution components to an initial empty solution, until a feasible solution is completed; and
- *Local Search* Heuristics: that repeatedly attempt to improve a current feasible solution by local changes.

A basic constructive heuristic is to add a solution component step by step in a greedy fashion as presented in Algorithm 1. For example, a popular example of a greedy constructive heuristic is the nearest-neighbour heuristic for the TSP. A solution x is built as follows. Function `InitialComponent` adds an initial random city as the first solution component, denoted as x_0. Then the nearest city, i.e., the city that has

Algorithm 2 Iterative Improvement Heuristic

```
1: $x \leftarrow$ GenerateSolution
2: repeat
3:     $\mathcal{N}(x) \leftarrow$ GenerateNeighbourhood($x$)
4:     $x' \leftarrow$ FindBest($\mathcal{N}(x)$)
5:     if ($f(x') < f(x)$) then
6:         $x \leftarrow x'$
7:     end if
8: until (no further improvement of $x$)
9: return $x$
```

not been selected yet and has the smallest distance from the current one, is selected which is defined by function `SelectGreedyComponent` and is added to x using the $\cup$ operator. This process is repeated until all cities are visited. A feasible solution is completed by adding at the end the first city of the solution.

A basic local search heuristic is to generate neighbourhood solutions for the initial solution, generated by function `GenerateSolution`, and search to find the best one from the neighbourhood to replace the initial solution, which is defined by function `GenerateNeighbourhood`. This process is repeated until no further improvement is found and it is often called iterative improvement or hill-climbing as presented in Algorithm 2. A popular example of an iterative improvement heuristic is the *k-exchange* for the TSP [164]. Given a solution, a 2-exchange neighbour solution is generated by exchanging two pairs of arcs. Then, a 2-exchange neighbourhood consists of solutions with all possible ways of exchanging arcs.

The former heuristics are able to provide a solution faster than local search heuristics, whereas the latter heuristics usually provide a better solution than the constructive heuristics. This is due to the generation of neighbourhood which usually requires a bit more computation time than adding solution components. Moreover, constructive heuristics generate a limited number of solutions since they only have a single searching iteration, whereas the local search heuristics constructs a neighbourhood of solutions that provide a searching mechanism. Hence, they are more likely to find a better solution than constructive heuristics, usually a local optimum solution. A disadvantage of a local search heuristic is that the algorithm may terminate in a poor-quality local optimum. A direct solution to this problem is to restart the local search and start from a new initial solution. From some experiments it was observed that the improvements by restarting the local search were not very significant [139].

3.4 Metaheuristics for Combinatorial Optimization

To avoid getting stuck in a local optimum solution and guide heuristics into promising areas in the search space to provide high quality solutions, *metaheuristics* have been widely used. Metaheuristics can be defined as a general algorithmic framework that can define problem-specific heuristic methods to a specific problem, and with few modifications to a wide set of different optimization problems.

Examples of metaheuristics that generalize the constructive and local search heuristics are: the iterated local search (ILS) [170], guided local search (GLS) [260] and the greedy randomized adaptive search procedures (GRASP) [85]. Their difference is in the way the two heuristic methods are activated since different algorithmic concepts are used. For example, GRASP consists of a constructive phase and a local search phase, where a greedy randomized solution is generated in which the local search is applied. ILS and GLS attempt to perform stochastic search in a space of the local optima with respect to some local search algorithms using the perturbed solution of the local optimum solution found previously as the new initial solution for the next perturbation iteration and using a dynamic objective function with penalties that are updated on every iteration, respectively.

Another type of metaheuristics is the nature-inspired algorithms that are inspired and use abstract models from natural phenomena or physical processes. Popular examples are EAs, simulated annealing (SA), tabu search (TS), estimatation of distribution algorithms (EDAs), bee colony optimization (BCO) and ACO. Since in this thesis we deal only with nature-inspired algorithms we describe them in detail in the following sections.

3.4.1 Evolutionary Algorithms

EAs refer to a class of three main algorithmic developments inspired from natural evolution: genetic algorithms (GAs) [129], evolutionary programming (EP) [89] and evolution strategies (ES) [240]. The main concept of EAs is the "survival of the fittest".

The common characteristics of these approaches are that they are population-based and iterative. Each individual (chromosome) in the population represents a potential solution to the optimization problem under consideration, and, thus, a specific point in the search space. The actual solution represented is otherwise known as the *genotype* of an individual. The individuals are able to exchange information using search operators, i.e., artificial *crossover* and *mutation*, and compete with each other using *selection*.

The selection operator considers the fitness (solution quality) of each individual, where the fittest ones are most possibly selected to generate the next generation. The fitness is otherwise known as the *phenotype* of an individual. The crossover operator considers two individuals and exchanges a certain amount of their existing content. In result, new individuals are generated, called the *offspring*. The mutation operator considers a single individual and modifies its contents randomly. Crossover and mutation operators are applied with probabilities p_c and p_m, respectively.

Algorithm 3 Evolutionary Algorithm

```
1: $t \leftarrow 0$
2: $P(0) \leftarrow$ InitializePopulation($\mu$)
3: $x^{bs} \leftarrow$ FindBest($P(0)$)
4: while (termination condition *not* met) do
5:     $P'(t) \leftarrow$ Selection($P(t)$)
6:     Crossover($P'(t)$, $p_c$)
7:     Mutation($P'(t)$, $p_m$)
8:     $x^{ib} \leftarrow$ FindBest($P'(t)$)
9:     if ($f(x^{ib}) < f(x^{bs})$) then
10:        $x^{bs} \leftarrow x^{ib}$
11:    end if
12:    $t \leftarrow t + 1$
13:    $P(t) \leftarrow P'(t-1)$
14: end while
15: return $x^{bs}$
```

The differences of different EAs lie on the representation of the individuals and the search operators used. For example, EP and ES use only the mutation operator and represent continuous variables, whereas GAs use both operators and represent discrete variables. Note that in this paper when we refer to EAs we consider the development of GAs since they are often used for COPs.

The general framework of EAs is represented in Algorithm 3, where $P(t)$ denotes the population of individuals at iteration t, μ the population size, and x^{bs} the best individual found so far. To define an EA the following functions have to be specified:

- `InitializePopulation`: generates the initial population.
- `FindBest`: returns the best individual in the current population for generation t, denoted as x^{ib}.
- `Selection`: selects individuals (parents) probabilistically, where the fittest ones have more chances to be selected.
- `Crossover`: combines parents by exchanging information to generate new individuals (offspring).

- `Mutation`: perturbs one individual randomly.

3.4.2 Simulated Annealing

SA is inspired by the physical annealing, which is the thermal process of solids to obtain lower energy state [146]. The solids are heated to increase their size and then the temperature is decreased very slowly, to obtain the perfect configuration by minimizing the internal energy state. Differently from EAs, SA is a single population algorithm.

The general idea of SA is to accept worse solutions under some criteria in order to escape a local optimum solution. Initially, the algorithm starts with a feasible solution x and at each iteration, a neighbour solution $x' \in \mathcal{N}(x)$ is generated that aims to replace x. If x' improves x then it is accepted, otherwise it is probabilistically accepted. The probability depends on the difference in the objective function of the two solutions, i.e., $f(x) - f(x')$, and on the temperature parameter $\mathcal{T}$. Inspired by the physical annealing process, $\mathcal{T}$ is lowered, and, thus, the probability of accepting a worse solution than the current one is reduced.

The probability p_{accept} to accept a solution x' when it is worse than the current one x is based on the Metropolis distribution [183], and it is defined as follows:

$$p_{accept}(x, x', \mathcal{T}) = \begin{cases} 1, & \text{if } f(x') < f(x), \\ \exp\left(\frac{f(x)-f(x')}{\mathcal{T}}\right), & \text{otherwise.} \end{cases} \tag{3.2}$$

The general framework of the algorithm is presented in Algorithm 4. To define a SA algorithm the following functions have to be specified:

Algorithm 4 Simulated Annealing Algorithm

```
1: $t \leftarrow 0$
2: $x \leftarrow$ GenerateSolution
3: $\mathcal{T} \leftarrow$ InitializeTemperature
4: $x^{bs} \leftarrow x$
5: while (external-loop termination condition *not* met) do
6:    while (internal-loop termination condition *not* met) do
7:       $x' \leftarrow$ GenerateNeighbour($x$)
8:       $x \leftarrow$ AcceptSolution($x$,$x'$,$\mathcal{T}$)
9:       if ($f(x) < f(x^{bs})$) then
10:          $x^{bs} \leftarrow x$
11:      end if
12:   end while
13:   $\mathcal{T} \leftarrow$ UpdateTemperature
14:   $t \leftarrow t + 1$
15: end while
16: return $x^{bs}$
```

- `InitializeTemperature`: assigns an initial value for the temperature parameter.
- `GenerateSolution`: generates the initial solution.
- `GenerateNeighbour`: chooses a new solution x' in the neighbourhood of solution x.
- `AcceptSolution`: uses Equation 3.2 to decide whether to accept or not solution x'.
- `UpdateTemperature`: returns a new value for $\mathcal{T}$.

3.4.3 Tabu Search

TS uses *memory structures* to guide the search process [117, 118]. It is not completely inspired from nature but it emulates the human problem solving process

Algorithm 5 Tabu Search Algorithm

```
1: $t \leftarrow 0$
2: $x \leftarrow$ GenerateSolution
3: InitliazeMemory
4: $x^{bs} \leftarrow x$
5: while (termination condition not met) do
6:     $\mathcal{N}'(x) \leftarrow$ GenerateModifiedNeighbourhood($x$)
7:     $x \leftarrow$ FindBest($\mathcal{N}'(x)$)
8:     UpdateMemory($t$)
9:     if ($f(x) < f(x^{bs})$) then
10:        $x^{bs} \leftarrow x$
11:    end if
12:    $t \leftarrow t + 1$
13: end while
14: return $x^{bs}$
```

[120]. TS has an intelligent search process and a flexible memory which restricts the search process to move back in any previously visited solution.

More precisely, TS modifies the neighbourhood $\mathcal{N}(x)$ of the current solution x to $\mathcal{N}'(x)$ in order to select elements included in $\mathcal{N}'(x)$ and not to select elements excluded when $\mathcal{N}(x)$ is modified into $\mathcal{N}'(x)$, where $\mathcal{N}'(x) \subseteq \mathcal{N}(x)$. In this way, it keeps a short-term history record of the moves performed during the search process. Moreover, $\mathcal{N}'(x)$ may be expanded to include additional solutions that are not found in $\mathcal{N}(x)$. In this way, it keeps a long-term history record [119].

In every iteration, TS makes the best possible move from x to a neighbour solution x' even if the new solution is worse than the current one. Then, the solution attributes of x' are declared as *tabu* to prevent moving back to them. The duration of solution attributes that are stored in the tabu list depends on the tabu list length, called *tabu tenure*. However, this may forbid moves toward attractive unvisited solutions, and, thus, *aspiration criteria* are used to override the duration of certain moves. A commonly used aspiration criterion is to allow tabu-*ed* solutions which are better than the current best solution.

The framework of simple TS algorithm is presented in Algorithm 5. To define a TS algorithm the following functions have to be specified:

- `GenerateSolution`: generates the initial solution.
- `InitializeMemory`: initializes the memory structure used as "tabu".
- `GenerateModifiedNeighbourhood`: generates a set of solutions, denoted as $\mathcal{N}'(x)$, that are not tabu or are tabu but satisfy the aspiration criterion.
- `FindBest`: returns the best solution in the current modified neighbourhood.
- `UpdateMemory`: updates the memory on iteration t.

3.4.4 Estimation of Distribution Algorithms

EDAs are considered as a special class of EAs [8, 151]. Similarly with other EAs, EDAs are iterative population-based algorithms, but search operators are not used. Instead, a probability distribution is learnt from a set of the fittest solutions, chosen by selection, in order to avoid explicit modelling. Offspring are constructed by sampling the probability distribution of the population.

Several EDAs with different probabilistic models have been proposed and are classified into three categories: *univariate*, *bivariate* and *multivariate*. Univariate EDAs assume that each variable in the solution is independent, whereas bivariate and multivariate EDAs assume pairwise interaction and more than two interactions among variables in the solution, respectively.

The framework of a simple EDA algorithm is presented in Algorithm 6, where $P(t)$ denotes the population of individuals at iteration t, μ the population size, and x^{bs} the best individual found so far. To define an EDA algorithm the following functions have to be specified:

Algorithm 6 Estimation Of Distribution Algorithm

```
1: $t \leftarrow 0$
2: $P(0) \leftarrow$ InitializePopulation($\mu$)
3: $x^{bs} \leftarrow$ FindBest($P(0)$)
4: **while** (termination condition *not* met) **do**
5:     Selection($P(t)$)
6:     $\mathcal{D}(t) \leftarrow$ GenerateProbabilisticDistribution
7:     $P'(t) \leftarrow$ SampleDistribution($\mathcal{D}(t)$)
8:     $x^{ib} \leftarrow$ FindBest($P'(t)$)
9:     **if** ($f(x^{ib}) < f(x^{bs})$) **then**
10:        $x^{bs} \leftarrow x^{ib}$
11:    **end if**
12:    $t \leftarrow t + 1$
13:    $P(t) \leftarrow P'(t-1)$
14: **end while**
15: **return** $x^{bs}$
```

- `InitializePopulation`: generates the initial population.
- `FindBest`: returns the best individual in the current population for generation t, denoted as x^{ib}.
- `Selection`: selects individuals (parents) probabilistically, where the fittest ones have more chances to be selected.
- `GenerateProbabilisticDistribution`: calculates the probabilistic model of the current population at iteration t, denoted $\mathcal{D}(t)$.
- `SampleDistribution`: generates a new population from the current probabilistic model.

3.4.5 Bee Colony Optimization

BCO is classified as a swarm intelligence (SI) algorithm, because it is inspired from the collective behaviour of agents in nature. In nature, scout bees leave their hive to explore sources of nectar, pollen and propolis. Then they return back to their hive and report to foraging bees about the quantity, quality and the location of the sources. Bees exchange information and can convince their nest mates to follow them by performing a *waggle* dance. When foraging bees decide to leave the hive they follow the trail of one of the dancing bees. In this way, they become committed followers of the dancer bees. Foraging bees return to the hive to pass the nectar to the food-storer bees. Then, they have three choices: (1) abandon the specific food source and become uncommitted followers; (2) follow the path for the specific food source again; and (3) become dance bees to recruit their nest mates. Hence, the richer the food source, the more bee recruitments [33].

BCO is an iterative population-based algorithm, in which every artificial bee generates a feasible solution for the optimization problem [169]. Initially, each bee represents an empty solution. Each bee consists of two passes, *forward* and *backward* pass. Each forward pass consists of a predefined number of n_c moves a bee can perform. Therefore, every bee constructs a partial solution in each move of the forward pass. Then, each forward pass corresponds to a backward pass, in which all bees share their information about the quality of their partial solutions. The bees decide, if they will become committed bees and drop their partial solution, or if they will become recruiters. It is obvious that the bees with better partial objective function have more chances to become recruiters, and, thus, follower bees inherit the partial solution of any recruiter bee, probabilistically. For example the partial objective function for the TSP, is the length of the partial tour. This process is repeated until a feasible solution is generated for all bees, and then the next iteration begins.

There are many algorithmic developments that use the behaviour of bees in the literature, proposed for different types of problems [169, 249, 268]. In this thesis, we consider the BCO development for COPs, which is in fact among the first developments of bee algorithms [169].

Algorithm 7 Bee Colony Optimization

```
1: $t \leftarrow 0$
2: $P(0) \leftarrow$ InitiliazePopulation($\mu$)
3: $x^{bs} \leftarrow$ *empty solution*
4: **while** (termination condition *not* met) **do**
5:     **while** (complete solution *not* constructed) **do**
6:         $P(t) \leftarrow$ ConstructPartialSolutions($n_c$)
7:         EvaluatePartialSolutions($P(t)$)
8:         LoyaltyDecision
9:         RecruitingProcess
10:    **end while**
11:    $x^{ib} \leftarrow$ FindBest($P(t)$)
12:    **if** ($f(x^{ib}) < f(x^{bs})$) **then**
13:        $x^{bs} \leftarrow x^{ib}$
14:    **end if**
15:    $t \leftarrow t + 1$
16: **end while**
17: **return** $x^{bs}$
```

The general framework of BCO is presented in Algorithm 7, where $P(t)$ denotes the population of individuals at iteration t, μ the population size, and x^{bs} the best individual found so far. To define a BCO algorithm the following functions have to be specified:

- `InitializePopulation`: generates the initial population.
- `ConstructPartialSolutions`: constructs a partial solution of n_c steps for each bee.
- `EvaluatePartialSolutions`: evaluates the partial solutions constructed in the forward pass and begin the backward pass.
- `LoyaltyDecision`: each bee decides probabilistically whether to become a recruiter bee and follow its own exploration or become an uncommitted bee and drop its partial solution.
- `RecruitingProcess`: each uncommitted bee decides probabilistically which recruiter bee to follow and inherit its partial solution.

- `FindBest`: returns the best bee solution in the current population for iteration t, denoted as x^{ib}.

3.4.6 Ant Colony Optimization

ACO is classified as a SI algorithm. The inspiration of ACO algorithms comes from the foraging behaviour of real ant colonies in nature; see Chapter 1. Since ACO is the main metaheuristic used in this thesis, it will be discussed in great detail in Chapter 4.

3.5 Characteristics of Metaheuristics

All metaheuristic algorithms have exploration and exploitation mechanisms. It is very important to achieve a good balance between the exploration and exploitation capacities in order for an algorithm to perform well for an optimization problem. In other words, an algorithm should be able to explore the search space for high quality or near the optimum solutions, and to exploit them. In case an algorithm performs too much exploration, it will lead the search process to randomization, whereas too much exploitation will lead the search process to converge in a poor quality solution.

Each metaheuristic algorithm is designed with different characteristics, where some of those characteristics achieve the exploration/exploitation balance. In Table 3.1, all the aforementioned metaheuristics for combinatorial optimization are classified according to their main characteristics.

Trajectory: Refers to the way metaheuristics move in the search space. Trajectory methods perform close *jumps* to the search space within the neighbourhood of a solution. Typical examples are SA and TS [255].

Population-based: Refers to the number of search points the metaheuristic consists. Population-based metaheuristics have multiple search points where each one is associated with an individual in the population. Typical examples are EAs, EDAs, ACO and BCO.

Global Optimizer: Refers to the way metaheuristics search. EAs, ACO, EDAs, and BCO are considered as global optimization algorithms because they can search in many points in the search space in parallel, whereas local optimizers, e.g., ILS, GLS, TS, SA etc., search only a single point in the search space. Often, global optimizers are applied with local search since they can provide good initial values for the local optimizers to improve the solution quality.

Memory-based: Refers to the search experience, in order to avoid moving to the same area in the search space. TS is the only metaheuristic that uses memory explicitly. EAs, EDAs, ACO and BCO algorithms have a kind of implicit memory, if we consider the solutions stored in the population or the pheromone trails, which are updated adaptively, on each iteration.

Probabilistic Vector: Refers to the way some metaheuristics construct their solutions. EDAs construct their solutions probabilistically on each iteration using a probabilistic vector based on the solution quality of the previous iteration, whereas the other metaheuristics have an actual solution, or solutions for EAs, that are updated or replaced by new ones generated under other considerations. ACO and BCO have this characteristic implicitly since each move of an ant depends on the existing amount of pheromone located on each link and each move of a bee depends on the existing number of visits bees made on each link, respectively. For instance, the pheromone trails can be seen as the probabilistic vector in ACO.

Dynamic Objective: Refers to the objective function a metaheuristic uses with respect to the optimization problem. GLS uses a *dynamic* objective function whereas

Table 3.1: Classification of the aforementioned metaheuristics according to their characteristics, where $\surd$, $\perp$ and χ indicates that the characteristic is present, partially present and not present, respectively.

Characteristic	EAs	SA	TS	EDA	ACO	BCO	ILS	GLS	GRASP
Trajectory	χ	$\surd$	$\surd$	χ	χ	χ	χ	χ	χ
Population-Based	$\surd$	χ	χ	$\surd$	$\surd$	$\surd$	χ	χ	χ
Global Optimizers	$\surd$	χ	χ	$\surd$	$\surd$	$\surd$	χ	χ	χ
Memory-Based	$\perp$	χ	$\surd$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	χ
Probabilistic Vector	χ	χ	χ	$\surd$	$\surd$	$\surd$	χ	χ	χ
Dynamic Objective	χ	χ	χ	χ	χ	χ	χ	$\surd$	χ
Nature-Inspired	$\surd$	$\surd$	$\perp$	$\surd$	$\surd$	$\surd$	χ	χ	χ

the remaining use a *static* objective function. GLS introduces penalties to the function that cause the objective to change.

Nature-Inspired: Refers to the inspiration of the metaheuristics. It is a minor characteristic, however it is important to understand the behaviour of the algorithm. EAs, ACO, BCO, EDAs and SA are inspired from nature, e.g., natural evolution, social insects, physical phenomena, etc. TS is partially inspired from nature, e.g, emulating the human problem solving process [120], whereas ILS, GLS, and GRASP are inspired by techniques based on the efficient solution of the problem.

3.6 Performance Analysis of Metaheuristics

The weak theoretical foundations of metaheuristics have been a major criticism for the evaluation concerning their performance, since there is no deep understanding why and how these algorithms work. More precisely, a theoretical analysis can guarantee the performance of an algorithm for all problem instances.

However, mathematically it is very challenging to provide a theoretical analysis for metaheuristics because they are complex algorithms and combine several methods in one. Convergence proofs for different metaheuristics have been proposed, but they are usually based on a very simple version of a metaheuristic, which is applied on a very simple optimization problem [83, 121, 236].

Due to the challenges regarding theoretical analysis, metaheuristics are usually analyzed and demonstrated empirically. Differently from theoretical analysis, empirical analysis is easier, since the implementation of an algorithm is not as challenging as theoretical analysis. However, it provides performance guarantees only for a limited number of problem instances, whereas theoretical analysis may cover all problem instances of an optimization problem.

The empirical analysis of metaheuristics usually takes into account two main aspects: the *effectiveness* and *efficiency* of the algorithm. The effectiveness is based on the solution quality, i.e., the best an algorithm can perform. The solution quality results are usually averaged out of several runs of the algorithm. The efficiency is based on the CPU-time, i.e., the computational effort of the algorithm. Therefore, for a fair comparison of the efficiency between algorithms, the same implementation style should be used.

In addition, further characteristics of the metaheuristics should be taken into account. For example, EAs and ACO algorithms perform several evaluations per iteration because they are population-based algorithms, whereas SA and TS perform a single evaluation per iteration. Therefore, each algorithm should perform the same number of function evaluations, in order to have a fair comparison among different algorithms. More precisely, in case an EA with a population size of μ individuals is compared with a SA algorithm, then a single iteration of an EA corresponds to μ iterations of the SA algorithm.

3.7 Summary

In this chapter we have presented the combinatorial optimization framework and the $\mathcal{NP}$-completeness theory that characterizes the difficulty of COPs. Different metaheuristics have been applied on COPs since they trade their optimality for efficiency. However, the theoretical work of metaheuristics is weak since they are complex algorithms and combine several methods. For this reason, metaheuristics are often analyzed empirically on existing benchmark problem instances, either with static or dynamic environments.

Several existing metaheuristics are explained, in this chapter, such as: EAs, SA, TS, EDAs, BCO, and ACO. Moreover, the aforementioned metaheuristics are classified according to their characteristics. More information for different metaheuristics and their applications can be found in [45].

In general, COPs are intriguing because they are easy to define but often difficult to solve. Moreover, many problems in the real-world can be modelled as COPs. Therefore, it is important to find or develop algorithms that solve them efficiently. The difficulty to solve most COPs is explained by the $\mathcal{NP}$-completeness theory, which classifies problems according to their difficulty.

Chapter 4

The Ant Colony Optimization Metaheuristic

4.1 Gentle Introduction to ACO

ACO algorithms are inspired from the foraging behaviour of real ant colonies as described in detail before in Chapter 1. In fact, this self-organizing behaviour is an example of *stigmergy* [254], where ants stimulate the remaining ants in the colony by modifying the environment via pheromone trail updating.

Moreover, the classical TSP was important for the development of the ACO framework. This is because it is the most fundamental $\mathcal{NP}$-complete COP that has many extensions in several applications. Also, it is a graph problem and, thus, ACO can be easily applied and understood; see Figure 2.6. In the next sections of this chapter we describe ACO with respect to the application on the TSP.

The general idea of ACO is to let ants walk on the links of cities in a graph G, and construct a feasible solution. Each ant has a memory that records the movements

Algorithm 8 Ant Colony Optimization for SOPs

```
1: $t \leftarrow 0$
2: $P(0) \leftarrow$ InitializePopulation($\mu$)
3: InitializePheromoneTrails($\tau_0$)
4: $x^{bs} \leftarrow$ *empty solution*
5: **while** (termination condition *not* met) **do**
6:     $P(t) \leftarrow$ ConstructAntSolutions
7:     PheromoneEvaporation($\rho$)
8:     DepositPheromone($P(t)$)
9:     $x^{ib} \leftarrow$ FindBest($P'(t)$)
10:    **if** $(f(x^{ib}) < f(x^{bs}))$ **then**
11:        $x^{bs} \leftarrow x^{ib}$
12:    **end if**
13:    $t \leftarrow t + 1$
14: **end while**
15: **return** $x^{bs}$
```

to the cities. The solution construction is based on existing pheromone trails and heuristic information. When all ants construct a feasible solution, pheromone is added to the path represented by each ant. In this way, the solutions are stored to the pheromone trails and considered for the construction phase of the next iteration. An indirect communication is achieved between the ants via their pheromone trails. Furthermore, a small amount, i.e., ρ, of pheromone is deducted from all the links of the cities, to help ants "forget" bad choices made in previous iterations.

The exploitation of ACO metaheuristic is achieved in the constructive phase with the consideration of heuristic information, e.g., in case of the TSP, it is the distance between the cities, whereas exploration is achieved by the pheromone trails. If ACO considers only heuristic information, the closest cities are more likely to be selected, which corresponds to a classic stochastic greedy algorithm. If ACO considers only pheromone information, a random tour is more likely to be selected because the existing pheromone trails are equal in the initial phase.

The general framework of the ACO metaheuristic for SOPs is presented in Algorithm 8, where $P(t)$ denotes the population of ants at iteration t, μ the population size, τ_0 is the initial pheromone value, and x^{bs} the best ant found so far. To define an ACO algorithm the following functions have to be specified:

- `InitializePopulation`: generates the initial population of ants.
- `InitializePheromoneTrails`: initializes all the trails with an equal value of pheromone.
- `ConstructAntSolutions`: each ant constructs a feasible solution.
- `PheromoneEvaporation`: all pheromone trails are reduced with a constant value.
- `DepositPheromone`: each ant deposits pheromone to mark its solution.
- `FindBest`: returns the best bee solution in the current population for iteration t, denoted as x^{ib}.

4.2 Characteristics and Properties

The ACO metaheuristic is categorized in the class of SI algorithms. Some researchers consider ACO as special class of EAs because it is believed that the communication ants have via their pheromone trails is evolution. In contrast, other researchers believe that ACO is not an EA because it does not have selection and crossover, and, thus, evolution is not achieved. However, EAs and ACO have similar characteristics since both of them are population-based, global optimizers and iterative. Moreover, they both use adaptive memory, if we consider the population of individuals on EAs and the pheromone table on ACO as memory that is updated adaptively.

If ACO algorithms and EDAs are taken into account and compared, it will be observed that their similarities are more than their differences. EDAs are considered as

EAs based on a probabilistic model. For example, the population-based incremental learning (PBIL) algorithm [8], an EDA, starts from an initial random population of individuals, the next population is generated based on the probabilistic vector. Then, the new population is evaluated and the probabilistic vector is modified accordingly. Similarly, in ACO the pheromone table constructed by the ants on each iteration has a similar role with the probabilistic vector in the PBIL, i.e., the pheromone trail values are the probabilities of selecting the next city to move. Their main difference is that in the PBIL all the components of the probability vector are evaluated independently. Both EDAs and ACO algorithms are constructive heuristics since the solutions are generated from scratch probabilistically, whereas on EAs the population exists and is evolved using the search operators.

Similarly with the other metaheuristics, it is very challenging to have theoretical analysis for ACO. A simple ACO metaheuristic, known as 1-ANT, has been used to provide theoretical proofs [148, 198]. 1-ANT assumes only a single ant and heuristic information is not considered. Hence, the theoretical ACO metaheuristic version has many differences from the practical ACO metaheuristic, which almost changes the general concepts of the traditional ACO framework.

4.3 Historical Contributions

In this section, the main contributions of ACO algorithms are reviewed starting from the first biological inspiration to the different applications regarding ACO.

- **1959**: The notion of stigmergy was described for the termites and defined as "stimulation of workers by the performance they have achieved" [104].
- **1983**: The probabilistic behaviour of ants was studied [54].

- **1988**: The collective and self-organizing behaviour of ants was studied [197].
- **1989**: The double bridge experiments of Argentine ant species *I. humilis* were performed [110].
- **1990**: The first probabilistic model was developed inspired from the behaviour of ants [53].
- **1991**: Three versions of Ant System (AS) were proposed called *ant-destiny*, *ant-quantity* and *ant-cycle* [67]. In the first two versions the ants update their pheromone directly after they choose one neighbour city, whereas in the last one the pheromone is updated after all ants constructed their solutions.
- **1993**: It was found that in the ant species *Lasius niger* the ants returning from rich food sources deposit more pheromone than those returning from poorer food sources [10].
- **1996**: The final version of the AS was developed for discrete optimization using the ant-cycle version due to its better performance than the other two versions [68].
- **1996**: The first application of ACO in telecommunication networks was developed [228].
- **1997**: Two of the best performing ACO algorithms were proposed (initially for the TSP), i.e., $\mathcal{MAX}-\mathcal{MIN}$ AS ($\mathcal{MM}$AS) [244] and Ant Colony System (ACS) [65]. In addition the first integration of ACO with a local search operator was proposed [244].
- **1998**: A multi-colony ACO was proposed where independent ant colonies run in parallel [242].
- **1999**: Stigmergy has been discussed for artificial ants in ACO [254].

- **1999**: A nondeterministic ACO algorithm was proposed, called approximate nondeterministic tree search [173].
- **1999**: The first ACO for multi-objective optimization was proposed [100].
- **2000**: A convergence proof for ACO was proposed [115].
- **2001**: The first real-world applications based on ACO metaheuristic[1] was developed.
- **2002**: The population-based ACO (P-ACO) was proposed, which is the memory-based version of traditional ACO [112].
- **2001**: The first ACO for dynamic optimization was proposed [114].
- **2002**: The first ACO for stochastic optimization was proposed [12].
- **2004**: The first ACO algorithm for continuous optimization was proposed [237].
- **2005**: An ACO algorithm was proposed to train Neural Networks [16].
- **2009**: A runtime analysis of a simple ACO algorithm was proposed [198].

Currently, most researchers of the ACO community are focused to the swarm robotics area using ideas from ants [61, 62]. Although much research has been done on different applications of ACO, most of it has been done considering stationary environments, whereas the research of ACO for DOPs is still in its infancy. However, DOPs are important problems to address, since they have many similarities to real-world situations, but they are also much more complex (more will be discussed in Chapter 6).

[1]All applications currently used in the industry are available at `www.antoptima.com`

4.4 Ant System (AS): The First ACO Algorithm

The AS consists of an initial phase and two iterative main phases, i.e., ants' solution construction and pheromone update [68]. In the initial phase, the pheromone trails are set with an equal amount τ_0, such that $\forall(i,j), \tau_{ij} = \tau_0 = \mu/C^{nn}$, where μ is the number of ants and C^{nn} is the cost of a greedy constructive heuristic described in Algorithm 1, e.g., for the TSP the nearest-neighbour heuristic is use, where a salesperson starts at a random city and repeatedly visits the nearest unvisited city until all have been visited.

In every iteration μ ants construct their solutions concurrently, each one starting from a randomly chosen city. Each ant k builds a solution city-by-city using a probabilistic decision rule, called *random proportional rule.* At each construction step the city selected is added to the partial solution of the ant. In particular, the probability that ant k chooses the next city j, when the last city in the partial tour is i, i.e., the current city, is defined as follows:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \qquad \text{if } j \in \mathcal{N}_i^k, \tag{4.1}$$

where τ_{ij} and η_{ij} are the existing pheromone trail and heuristic information available a priori, respectively, α and β are the constant parameters that determine the relative influence of pheromone trail and the heuristic information, respectively, and $\mathcal{N}_i^k$ is the neighbourhood of unvisited cities of ant k when its current city is i. For the case of the TSP the heuristic information is defined as $\eta_{ij} = 1/d_{ij}$, which is inversely proportional to the the distance, i.e., d_{ij}, between cities i and j.

After all ants build a feasible solution T^k the pheromone trails are updated. At the beginning all the pheromone trails are lowered by a constant factor, due to the pheromone evaporation, such that:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \quad \forall (i,j) \in E, \tag{4.2}$$

where $0 < \rho \leq 1$ is the pheromone evaporation rate, which helps the ants to eliminate pheromone trails that are not used frequently and have been created from bad decisions previously taken. After evaporation, all ants deposit pheromone on the arcs of their path as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{\mu} \Delta\tau_{ij}^{k}, \quad \forall (i,j) \in E, \tag{4.3}$$

where $\Delta\tau_{ij}^{k}$ is the amount of pheromone ant k deposits on the arcs that belong to its tour T^k and is defined as follows:

$$\Delta\tau_{ij}^{k} = \begin{cases} 1/C^k, & \text{if } (i,j) \in T^k, \\ 0, & \text{otherwise}, \end{cases} \tag{4.4}$$

where C^k is the tour cost of the tour T^k constructed by ant k. As a result, the amount of pheromone of each ant is proportional to the solution quality. Hence, the better the ant's tour, the more pheromone an ant deposits.

4.5 AS Variations

After the development of the first ACO algorithm, i.e., AS, several variations have been proposed that mainly differ in the pheromone update. Moreover, an important extension has been proposed which has more differences than similarities from the AS. The main variations and the extension of AS are discussed in the next subsections.

4.5.1 Elitist AS

The Elitist AS (EAS) uses an *elitist strategy* where the best ant deposits an additional pheromone to its tour [68]. The initial phase and the solution construction phase are the same as in the AS algorithm. However, after pheromone evaporation, all the ants deposit pheromone as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{\mu} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{bs}, \quad \forall(i,j) \in E, \tag{4.5}$$

where $\Delta\tau_{ij}^{k}$ is defined as in Equation 4.4, e is the parameter that determines the influence of the elitist strategy and $\Delta\tau_{ij}^{bs}$ is defined as follows:

$$\Delta\tau_{ij}^{bs} = \begin{cases} 1/C^{bs}, & \text{if } (i,j) \in T^{bs}, \\ 0, & \text{otherwise,} \end{cases} \tag{4.6}$$

where C^{bs} is the tour cost of tour T^{bs} which is constructed by the *best-so-far* ant. Note that the best-so-far ant is a special ant that may not belong to the population in every iteration.

4.5.2 Rank-Based AS

In the *rank-based* AS (AS$_{rank}$) each ant deposits an amount of pheromone proportional to its rank [29]. In addition, the best-so-far ant always deposits a higher amount of pheromone than the other ants as in EAS. The initial phase and the solution construction are the same as in the AS algorithm. However, after pheromone evaporation all the ants are ranked according to their solution quality such that only the $w-1$ best-ranked ants and the best-so-far ant are allowed to deposit pheromone.

The best-so-far ant receives the highest weight to deposit pheromone, i.e., w, whereas the r-th best ant of the iteration has weight $\max\{0, w-r\}$. Formally, the pheromone update in AS$_{rank}$ is defined as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{r=1}^{w-1} (w-r)\, \Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs}, \tag{4.7}$$

where $\Delta\tau_{ij}^{bs}$ is defined as in Equation 4.6 and $\Delta\tau_{ij}^{r} = 1/C^{r}$ where C^{r} is the tour cost of T^{r} of the $r-$th best-ranked ant.

4.5.3 $\mathcal{MAX} - \mathcal{MIN}$ AS

The $\mathcal{MM}$AS is one of the best performing and well-studied ACO algorithms [244, 245]. Differently from the previous AS variations, in the $\mathcal{MM}$AS only either the best-so-far ant or the iteration-best ant is allowed to deposit pheromone. The initial phase and the solution construction phase are the same as in the AS algorithm, whereas the pheromone update is defined as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best}, \quad \forall (i,j) \in T^{best}, \tag{4.8}$$

where $\Delta\tau_{ij}^{best} = 1/C^{best}$. In case the best-so-far ant is allowed to deposit pheromone $\Delta\tau_{ij}^{best} = 1/C^{bs}$, whereas in case the iteration-best ant is allowed to deposit pheromone $\Delta\tau_{ij}^{best} = 1/C^{ib}$, where C^{ib} is the tour cost of the best ant of the current iteration.

Moreover, the pheromone trails are bounded to the interval $[\tau_{min}, \tau_{max}]$, where τ_{min} and τ_{max} are the lower and upper limits, respectively. Since only the best ant is allowed to deposit pheromone, high intensity of pheromone may be generated to a, suboptimal, solution. Hence, the mechanism that restricts the range of the pheromone trails avoids stagnation behaviour. Finally, in case of search stagnation or if for a given number of algorithmic iterations no improvement is found the pheromone trails are re-initialized to an estimate of the upper pheromone trail limit to increase exploration.

4.5.4 Best-Worst AS

In the Best-Worst AS (BWAS) only the best-so-far and the iteration-worst ants update their pheromone trails positively and negatively, respectively [42, 43]. The initial phase and the construction of solutions phase are the same as in the AS algorithm, whereas the pheromone update is defined as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{bs}, \quad \forall (i,j) \in T^{bs}, \tag{4.9}$$

where $\Delta\tau_{ij}^{bs}$ is defined as in Equation 4.6. In addition, the arcs that belong to the iteration-worst ant are penalized if they are not present in T^{bs}, such that:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \quad \forall (i,j) \in T^{iw}, \, (i,j) \notin T^{bs}, \tag{4.10}$$

where T^{iw} is the solution of the iteration-worst ant.

BWAS uses pheromone re-initialization slightly differently from the $\mathcal{MM}$AS, where all pheromone trails are set to the initial pheromone value τ_0 only if for a given number of algorithmic iterations no improvement is found.

Furthermore, BWAS uses concepts from evolutionary computation and introduces *pheromone mutation* to enhance the exploration. Therefore, in every iteration t the pheromone trails are mutated with probability p_{phmut} as follows:

$$\tau_{ij} = \begin{cases} \tau_{ij} + \tau_{mut}, & \text{if } R = 0, \\ \tau_{ij} - \tau_{mut}, & \text{if } R = 1, \end{cases} \tag{4.11}$$

where R is a random value in $\{0,1\}$ and τ_{mut} is the added/subtracted value of the pheromone trails, defined as:

$$\tau_{mut} = \frac{t - t_r}{G - t_r}\sigma\tau_{avg}, \tag{4.12}$$

where t is the current iteration, t_r is the iteration count of the last iteration a restart is performed, G is the maximum number of iterations, τ_{avg} is the average of the pheromones that exist on the arcs that belong to T^{bs}, and σ is a parameter that determines the influence of mutation. Note that in case τ_{ij} becomes negative from the subtraction in Equation 4.11 is set to small constant value, i.e., c_{min}.

4.5.5 Ant Colony System

ACS is an extension than a variation of AS, because their differences are not only in the pheromone update policy [65, 66]. ACS uses a more aggressive decision rule than AS that provides strong exploitation. More precisely, an ant k selects the next city j when it is located in city i as follows:

$$j = \begin{cases} \max_{l \in \mathcal{N}_i^k}\{\tau_{il}[\eta_{il}]^{\beta}\}, & \text{if } R \leq q_0, \\ \mathcal{J}, & \text{otherwise,} \end{cases} \tag{4.13}$$

where R is a random number in the interval $[0,1]$, $0 \leq q_0 \leq 1$ is a parameter of the decision rule, called *pseudorandom proportional rule*, and $\mathcal{J}$ is the random proportional rule defined in Equation 4.1. In other words, with probability q_0 ants make the best possible move, whereas with probability $(1-q_0)$ they make a move probabilistically.

As the ants build their solution they update the pheromone trails locally on the arcs. For instance, if an ant moves from city i to city j the pheromone trails of arc

(i, j) are updated directly as follows:

$$\tau_{ij} \leftarrow (1 - \xi)\tau_{ij} + \xi\tau_0, \tag{4.14}$$

where $0 < \xi < 1$ is a constant parameter and τ_0 is the initial pheromone value. When all ants construct their solution, the best-so-far ant is allowed to deposit pheromone as follows:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \rho\Delta\tau_{ij}^{bs}, \quad \forall (i, j) \in T^{bs}, \tag{4.15}$$

where ρ is the evaporation rate which is performed only to the pheromone trails that belong to the best-so-far solution T^{bs}, and Δ_{ij}^{bs} is defined as in Equation 4.6. It has been verified that the ACS has pheromone trail limits as with the $\mathcal{MMAS}$. The pheromone trails are implicitly bounded in $[\tau_0, (1/C^{bs})]$, different from $\mathcal{MMAS}$ where they are explicitly bounded.

Finally, ACS was the first algorithm that restricts the neighbourhood of unvisited cities for ants using *candidate lists*. In the AS and its variations $\mathcal{N}_i^k$ was defined as all the cities that have not been visited by ant k yet. In ACS, the unvisited cities are ranked with respect to the heuristic information η, and $\mathcal{N}_i^k$ contains only the cl best-ranked cities. Nowadays, most ACO algorithms use the restricted neighbourhood because it leads to better results regarding solution quality and efficiency (especially on large problem instances).

4.6 Applications of ACO for Combinatorial Optimization

The ACO metaheuristic has a wide range of applications in combinatorial optimization. Table 4.1 shows the important, or the most popular, ACO algorithms designed

Table 4.1: ACO applications for stationary combinatorial problems categorized by their type and sorted chronologically. The table updates and extends the applications given in [69, p. 39] and [64].

Problem type	Problem name	Year	Main References
Routing	Travelling Salesperson	1991	[67]
		1992	[46]
		1995	[97]
		1996	[68, 98]
		1997	[29, 66, 244]
		2000	[43, 144, 152, 245]
		2002	[112, 143]
		2004	[37, 152]
		2007	[213]
	Vehicle Routing	1997	[27]
		1999	[28, 100]
		2002	[57, 216, 219]
		2003	[217]
		2004	[11, 58, 218]
		2006	[158]
		2008	[63, 93]
		2009	[91, 94]
	Sequential Ordering	2000	[99]
Scheduling	Job-shop	1994	[47]
		2004	[296]
		2005	[133]
		2007	[298]
		2008	[9, 59]
		2009	[88, 165]

Table 4.1: (continued)

Problem type	Problem name	Year	Main References
		2010	[230, 266]
	Floor-shop	2002	[259]
	Flow-shop	1998	[243]
		2006	[95]
		2009	[165]
	Group-shop	2002	[14]
	Total Tardiness	1999	[7]
	Total Weight Tardiness	2000	[52, 184]
		2009	[165]
		2001	[185]
		2002	[92]
	Project-scheduling	2002	[186]
		2007	[231]
		2009	[292, 299]
	Open-shop	1996	[211]
		2005	[15]
Assignment	Quadratic Assignment	1994	[177]
		1999	[101, 173, 175]
		2000	[245]
		2001	[248]
		2002	[44]
		2006	[56]
		2008	[191]
	Course Timetabling	2002	[238]
		2003	[239]

Table 4.1: (continued)

Problem type	Problem name	Year	Main References
		2005	[70]
		2006	[77]
		2007	[78]
		2012	[199]
	Graph Colouring	1997	[48]
		2005	[227]
	Generalized Assignment	2002	[171]
	Frequency Assignment	2002	[174]
Subset	Bus-stop Allocation	2001	[51]
	Redundancy Allocation	1999	[159]
		2004	[160]
	Covering Problems	2000	[2]
		2004	[232]
		2011	[140]
		2002	[176, 215]
	Multiple Knapsack	1999	[153]
		2005	[86]
		2006	[149]
		2008	[87, 150]
		2010	[145]
	Max Independent Set	2001	[154]
	Maximum Clique	2003	[84]
		2007	[267]
Other	Shortest Common Sequence	1998	[187]
	Maximum-Cut	2008	[96]

Table 4.1: (continued)

Problem type	Problem name	Year	Main References
	Bin Packing	2004	[155]
	Data Mining	2002	[206]
	Protein folding	2003	[233]
	Water Supply Networks	2005	[1]
		2008	[167, 203]
		2010	[39]

for different applications with stationary environments. The applications are categorized to three main types: routing, scheduling and assignments. Of course, other types of applications exist. However, in this thesis we consider the application of ACO for the TSP and VRP which are classified in the routing category.

4.6.1 Travelling Salesperson Problem (TSP)

The TSP is one of the most popular and fundamental COPs. The objective of the TSP is to determine the permutation of cities that minimizes the length of a cyclic tour which contains each city once and only once. Figure 4.1 illustrates the optimal solution of a TSP instance with 13509 cities.

Formally, the TSP can be described as follows:

$$f(x) = \min \sum_{i=0}^{n} \sum_{j=0}^{n} d_{ij}\psi_{ij}, \tag{4.16}$$

subject to:

$$\psi_{ij} = \begin{cases} 1, & \text{if } arc(i,j) \text{ is in the tour,} \\ 0, & \text{otherwise,} \end{cases} \tag{4.17}$$

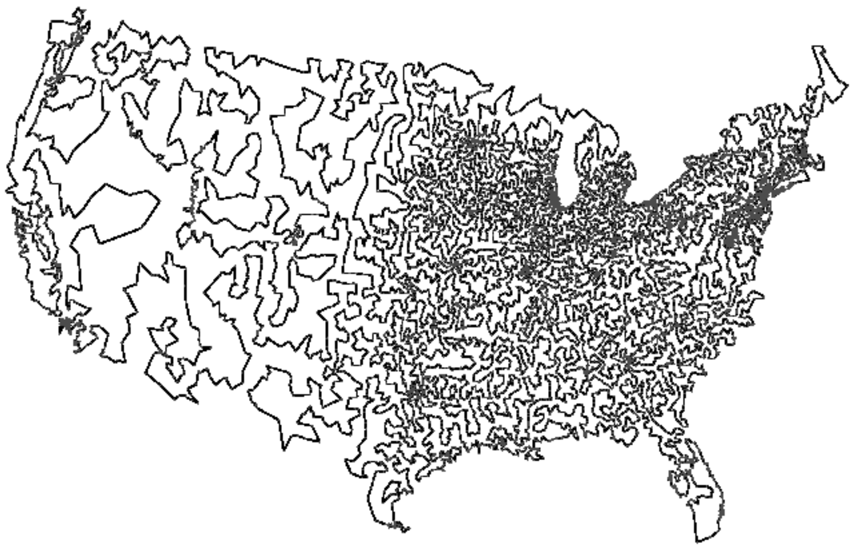


Figure 4.1: Example of a TSP solution of the usa13509t.tsp benchmark problem instance (Taken from: `http://www.tsp.gatech.edu/gallery/itours/usa13509.html`)

where $\psi_{ij} \in \{0, 1\}$, n is the number of cities, d_{ij} is the distance between city i and j.

The main existing applications of ACO regarding the TSP have been extensively described in Section 4.5, since all the variants of AS, i.e., EAS, AS$_{rank}$, $\mathcal{MM}$AS, BWAS, and ACS have been proposed for the TSP. Other improvements based on these algorithms [163, 207, 246, 252, 284] and hybridizations [60, 178, 295, 297] have been proposed. Below we briefly summarize the application of ACO for the TSP.

Structure. The TSP can be modelled directly using the problem graph $G = (V, E)$. Each city, or component of the permutation solution, corresponds an element of the set V. The links between the cities, correspond to a connection in the set E and the distance between cities correspond to the distance d_{ij} between cities i and j.

Constraints. The constraints of TSP are that all cities have to be visited once and only once and the last city in the permutation should be identical with the first one. This corresponds to return home of the salesperson after visiting the customers.

Solution construction. Initially, each ant is placed on a randomly selected city and iteratively adds one unvisited city, considering its neighbourhood $\mathcal{N}_i^k$, to its partial solution. In this way, the constraint of visiting each city once is satisfied. The construction of solution terminates when all cities are visited once. Then, the first component of the solution is added to the end to satisfy the second constraint. AS, EAS, AS$_{rank}$, $\mathcal{MM}$AS, and BWAS use the decision rule in Equation 4.1, whereas ACS uses the decision rule in Equation 4.13.

Pheromone trails and heuristic information. The pheromone trails τ_{ij} and heuristic information η_{ij} are associated to the links of cities i and j. The heuristic information is defined as $\eta_{ij} = 1/d_{ij}$ for all the algorithms.

Pheromone Update. The pheromone update procedure varies for all algorithms. In AS, all the ants deposit pheromone, whereas on the remaining algorithms different elitism strategies are used.

4.6.2 Vehicle Routing Problem (VRP)

The VRP is a more realistic variant of the TSP [50]. The VRP can be described as a number of vehicles with a fixed capacity needing to satisfy the demand of all the customers, starting from and finishing at the depot. Hence, a VRP without the capacity constraint or with one vehicle can be seen as a TSP. Figure 4.2 illustrates the optimal solution of a VRP instance with 53 customers served by 7 vehicles (routes).

Formally, the basic VRP, known as the capacitated VRP (CVRP), can be described as follows:

$$f(x) = \min \sum_{i=0}^{n} \sum_{j=0}^{n} d_{ij} \sum_{k=1}^{v} \psi_{ij}^{k}, \tag{4.18}$$

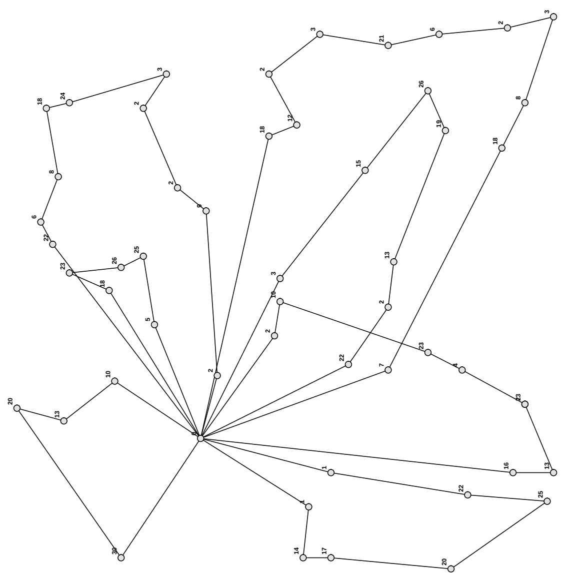


FIGURE 4.2: Example of a VRP solution of the A-n53-k7.vrp benchmark problem instance (Taken from: http://neo.lcc.uma.es/radi-aeb/WebVRP/data/instances/Augerat/A-opt-sol-graph.zip)

subject to:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\psi_{ij}^{k}\delta_i \leq Q^k, \forall k \in \{1,\ldots,v\}, \tag{4.19}$$

$$\psi_{ij}^{k} = \begin{cases} 1, & \text{if } arc(i,j) \text{ is covered by vehicle } k, \\ 0, & \text{otherwise,} \end{cases} \tag{4.20}$$

where $\psi_{ij}^{k} \in \{0,1\}$, n is the number of customers, v is the number of vehicles, δ_j is the demand of customer j and Q^k is the capacity of vehicle k.

Apart from the capacity constraint, other possible constraints can be imposed to the CVRP taken from real-world applications, such as service time, time windows, backhauls, multidepots, fuel availability, etc. Hence, VRP has many variations, to which some ACO algorithms have been applied, e.g., the VRP with time windows [63, 100], the VRP with backhauls [93, 219], the VRP with time windows and backhauls [217], the generalized VRP where the customers are clustered [210], the open VRP where the vehicles do not necessarily have to return to the depot [158], the VRP with simultaneous delivery and pickup [94] and the VRP with the two-dimensional load constraint [91]. In fact, the ACO metaheuristic is used in real-world applications related to the VRP (more details in [102, 224]).

In this thesis, we are focused on the applications of ACO algorithms regarding the basic CVRP, described in Equations 4.18, 4.19 and 4.20. Popular examples such as the AS$_{rank}$-CVRP [27, 28], Saving-based AS (SbAS) [217, 219] and multi-colony ACS-VRP (MACS-VRP) [100] are described below. There are also parallel implementations of these algorithms with promising results [11, 58].

Structure. The VRP can be modelled directly using the problem graph $G = (V, E)$. Each customer, or component of the permutation solution, including the depot component, corresponds to one component of the set V. The links between the cities, correspond to a connection to the set E and the distance between cities corresponds to the distance d_{ij} between cities i and j. Furthermore, a demand δ_i is associated with each customer i. This model is adopted by AS$_{rank}$-CVRP and SbAS. On the other hand, MACS-VRP uses the same model but it consists of multiple copies of $v - 1$ depots, called *dummy depots*, where v denotes the number of vehicles. The distances between the dummy depot components are zero since they have the same location.

Constraints. The constraints in the CVRP are that all customers have to be served once and only once by any vehicle, the vehicle must not exceed the capacity, and each vehicle must start and return to the depot.

Solution construction. In AS$_{rank}$-CVRP, the ants are placed on the depot and build a solution by selecting the next customer using the decision rule in Equation 4.1. In case the next customer selected violates the capacity constraint, then the current vehicle route is closed by the vehicle returning to the depot (without adding the customer selected to the route). A new vehicle route is started if there are any other unvisited customers. MACS-VRP consists of two interacting colonies, ACS-VEI and ACS-TIME, where the former one minimizes the number of vehicles and the latter one minimizes the travel time for a given number of vehicles. Each ant, from both colonies, starts from a randomly chosen dummy depot and builds a solution without violating the capacity constraint as in AS$_{rank}$-CVRP, using the decision rule in Equation 4.13. However, in MACS-VRP, an ant is allowed to return to the depot anytime, even when the capacity constraint is not violated by a customer. This kind of solution construction may generate infeasible solutions where not all customers are satisfied, due to the limited number of dummy depots that represent the number of vehicles. Therefore, a repair method is used to insert the unscheduled customers, if possible, at a position such that the travel time is minimized. In SbAS, the solution construction is based on the saving algorithm [40, 205], in which each customer is assigned as a separate tour and tours are combined as long as the capacity constraint is not violated, and until no more combinations are feasible. The combinations of tours are based on the saving heuristic function which is described below in Equation 4.22. Differently, SbAS uses the decision rule, in Equation 4.1, and the ants select probabilistically the next tours to be combined.

Pheromone trails and heuristic information. The pheromone trails τ_{ij} and heuristic information η_{ij} are associated to the links of cities i and j for the AS$_{rank}$-CVRP and the MACS-VRP, whereas for SbAS they are associated with the pairs of the customers generated initially. In AS$_{rank}$-CVRP, the heuristic information is based on the parametrized savings function [205] and defined as:

$$\eta_{ij} = d_{i0} + d_{0j} - g d_{ij} + f|d_{i0} - d_{j0}|, \tag{4.21}$$

where d_{i0} is the distance from customer i to the depot and f and g are two constant parameters (usually $g = f = 2$). In SbAS, the heuristic information is based on the standard saving function [40] and defined as:

$$\eta_{ij} = d_{i0} + d_{0j} - d_{ij}. \tag{4.22}$$

In MACS-VRP, the heuristic information is defined as a function of the travel time between two customers, of the number of times a customer was not included in ant's "infeasible" solution in previous iterations, and of other factors based on some constraints of the problem (for more details see [100]).

Pheromone update. The pheromone update procedures for AS$_{rank}$-CVRP and SbAS are based on AS$_{rank}$ defined in Equation 4.7. In MACS-VRP, both colonies are based on the ACS defined in Equations 4.14 and 4.15, and pheromone trails are updated according to their objectives.

4.7 Summary

In this chapter we have presented the ACO framework. ACO lies in the category of metaheuristics. It was inspired by the foraging behaviour of real ants. Since the development of the first ACO algorithm for the TSP, the area, regarding both algorithmic variations and applications, has grown rapidly. The main application and algorithmic contributions of ACO for SOPs are defined in this chapter. More details are given for the applications of TSP and VRP which are the problems used for the experiments in Chapters 5.

ACO showed good performance in many $\mathcal{NP}$-complete COPs, especially in problems where heuristic information is available, such as the TSP and its variations. In such problems, ACO has a slight advantage over other metaheuristics since it has prior knowledge of the problem instance. For example, in the TSP or VRP, ants consider the distances between cities or customers, while the ants construct their solutions.

In case heuristic information is not available in a COP, the integration of ACO with a local search is vital for good performance.

Similarly with other metaheuristics, ACO has a high risk to get stuck on a local optimum solution, especially as the problem size increases. Many modifications based on the pheromone update have been developed to improve the performance of ACO, with the aim of a better balance between exploration and exploitation.

Chapter 5

ACO with Direct Communication

5.1 Motivation

Generally, conventional ACO algorithms suffer from the stagnation behaviour, where all ants follow the same path from the initial stages of the algorithm, when applied to different combinatorial problems. This is because a high intensity of pheromone is generated to a single trail, that marks a potential solution, and attracts the ants to that area. Therefore, conventional ACO algorithms are more likely to get trapped in a local optimum solution, which may degrade the solution quality and the overall performance of the algorithm.

Different pheromone strategies have been applied to conventional ACO algorithms in order to increase exploration and avoid stagnation behaviour. Hence, many variations of the first ACO algorithm, e.g, AS, exist.

One advantage that ACO algorithms have over other metaheuristics when applied to the TSP and the VRP, is the heuristic information used in the solution construction. Such information enables the population to have prior knowledge of the problem instance before execution. Such knowledge is gained in other metaheuristics using local search (LS) operators to improve the solution quality. Similarly, LS operators have been integrated with ACO and improved its solution quality significantly [245]. In fact, on COPs where heuristic information is not available ACO has more random behaviour because it considers only pheromone trails, unless it is integrated with a LS operator.

5.2 Description of Direct Communication

5.2.1 The Framework

In nature, ant colonies do not only communicate indirectly via their pheromone trails, but also directly by exchanging important information, like sounds, displays or movements [132]. This extra communication the ants have may help them to improve their food searching tasks. In the case of ACO algorithms, a direct communication (DC) scheme may help the population of ants to avoid stagnation behaviour. Therefore, the proposed DC scheme allows ants to exchange variable objects, e.g., cities for the TSP or customers for the VRP, after they construct their solution and before they deposit pheromone as shown in Algorithm 9.

The DC scheme can be applied to any variation of ACO for the TSP or the VRP. Therefore, the solution construction and pheromone update policies depend on the ACO variation in which DC is applied.

It is possible that the solution quality of an ant may be worse than the best ant, but a subtour may belong to the global optimum. The aforementioned scheme may take advantage of the different solutions constructed by ants on each iteration with the information exchanged by the swaps.

Algorithm 9 Ant Colony Optimization with DC

```
1: $t \leftarrow 0$
2: $P(0) \leftarrow$ InitializePopulation($\mu$)
3: InitializePheromoneTrails($\tau_0$)
4: $x^{bs} \leftarrow$ *empty solution*
5: **while** (termination condition *not* met) **do**
6:     $P(t) \leftarrow$ ConstructAntSolutions
7:     $P'(t) \leftarrow$ *DirectCommunication* using Algorithm 10
8:     PheromoneEvaporation($\rho$)
9:     DepositGlobalPheromone($P'(t)$)
10:    $x^{ib} \leftarrow$ FindBest($P'(t)$)
11:    **if** ($f(x^{ib}) < f(x^{bs})$) **then**
12:        $x^{bs} \leftarrow f(x^{ib})$
13:    **end if**
14:    $t \leftarrow t + 1$
15: **end while**
16: **return** $x^{bs}$
```

5.2.2 Exchange Information

The exchange information process consists of two types of swaps, where with a higher probability ants swap objects obtained from ants within its communication range, and with a lower probability ants swap objects randomly as shown in Algorithm 10. At each step of the DC scheme, an object is selected and the successor and predecessor objects are swapped with others only if the distance between the object and the new selected object is better than that to the successor or the predecessor.

The adaptive swaps help to avoid randomization and model DC between ants, whereas random swaps help to avoid local optima, when adaptive swaps become ineffective. In fact, random and adaptive swaps have a similar effect in the DC scheme with the mutation and crossover operators in EAs (see Algorithm 3 for more details).

In the case of the VRP, the above swaps may violate the capacity and service time constraints of the vehicle routes and generate an infeasible solution. Therefore, only the swaps that do not violate any of the constraints are allowed.

Algorithm 10 DirectCommunication

```
1: for ($k = 0$ to $\mu$) do
2:     $o_i \leftarrow$ random object from ant $k$
3:     $s_i \leftarrow$ select successor of $o_i \in k$
4:     $p_i \leftarrow$ select predecessor of $o_i \in k$
5:     if ($rand[0.0, 1.0] \leq 0.2$) then
6:         $s_i' \leftarrow$ random object from ant $k$
7:         $p_i' \leftarrow$ random object from ant $k$
8:     else
9:         $k' \leftarrow$ select ant from the communication range of ant $k$ using Equation 5.1
10:        locate object $o_i \in k'$
11:        $s_i' \leftarrow$ select successor of $o_i \in k'$
12:        $p_i' \leftarrow$ select predecessor of $o_i \in k'$
13:        locate $s_i'$ and $p_i' \in k$
14:    end if
15:    if ($d(o_i, p_i) > d(o_i, p_i')$ && $k$ is *feasible* after swapping) then
16:        swap $p_i$ with $p_i' \in k$
17:        DepositLocalPheromone($o_i, p_i'$)
18:    end if
19:    if ($d(o_i, s_i) > d(o_i, s_i')$ && $k$ is *feasible* after swapping) then
20:        swap $s_i$ with $s_i' \in k$
21:        DepositLocalPheromone($o_i, s_i'$)
22:    end if
23: end for
```

5.2.2.1 Random Swaps

When the population of ants reaches stagnation behaviour, the information adapted by other ants will be identical with the current one. Therefore, no swaps will be performed using adaptive swaps and the adaptive swaps become ineffective. Random swaps can then move the population to another area in the search space and help to escape local optima.

In random swaps, the successor and predecessor objects to be swapped are selected randomly from the current ant and they are not adapted from any other ant. In a way, adaptive and random swaps act as crossover and mutation search operators in EAs, respectively, but for the ACO case. An example of an adaptive and a random swap is illustrated in Figure 5.1.

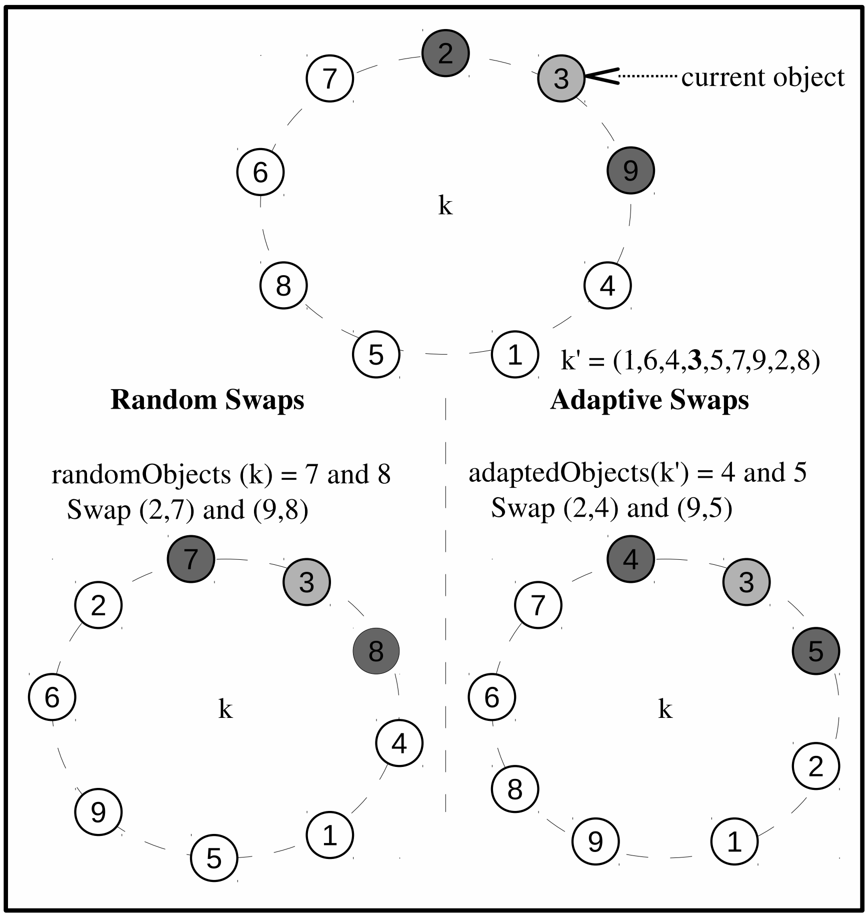


Figure 5.1: Example of adaptive and random swaps for the TSP.

5.2.2.2 Adaptive Swaps

Each ant k communicates with another ant within its communication range as follows:

1. An object o_i is randomly selected from ant k.
2. The successor and predecessor of o_i, i.e., objects s_i and p_i, respectively, are selected from ant k.
3. Another ant is randomly selected, i.e., ant k', from the communication range of ant k and the corresponding object o_i is located in ant k'.

4. The successor and predecessor of o_i, i.e., objects s'_i and p'_i, respectively, are selected from ant k'
5. Then, objects s'_i and p'_i are located in ant k
6. Swaps are performed in ant k between the objects s_i and s'_i and between objects p_i and p'_i
7. Moreover, a small extra amount of pheromone is deposited to the resulted edges between o_i and its new successor and between o_i and its new predecessor in ant k.

5.2.3 Communication Range

The communication range of each ant, say ant p, which determines its neighbour ants, is based on the similarities of ants and is defined as follows:

$$R_p = \{q \in P(t) | S(p,q) \leq T_r\}, \tag{5.1}$$

where $P(t)$ is the population of ants at iteration t, T_r is a predefined threshold which determines the size of the communication range of ant p, and $S(p,q)$ is the similarity metric between ants p and q. For the TSP, $S(p,q)$ is defined as:

$$S(p,q) = 1 - \frac{c_{E_{pq}}}{n}, \tag{5.2}$$

where $c_{E_{pq}}$ are the common edges of the objects between the two ants, and for the VRP it is defined as:

$$S(p,q) = 1 - \frac{c_{E_{pq}}}{n + avg(n_{V_p}, n_{V_q})}, \tag{5.3}$$

where n_{V_p} and n_{V_q} are the number of vehicles used in the solutions of ant p and q, respectively.

A value $S(p, q) = 1.0$ in both Equations 5.2 and 5.3 for TSP and VRP, respectively, denotes that the two ants are completely different, whereas a value $S(p, q) = 0.0$ denotes that the two ants are identical.

5.2.4 Local Pheromone Deposit

The above communication scheme may provide more exploration but it has a high risk to degrade the solution quality of the tours constructed by the ants and disturb the optimization process. Therefore, only the swaps which provide improvement are allowed in order to limit the risk. For example, if the distance d_{ij} between the current successor object of o_i in ant k is less than that to the successor city obtained from the neighbour ant k', the ant k remains unchanged. The same happens with the predecessor object. A similar swap method based on the position of the cities has been used in a discrete version of particle swarm optimization on the TSP [262]. However, in the adaptive swaps the positions of the swapped objects are not predefined but they are adapted from other ants in the population.

Apart from the swaps, a small amount of pheromone is deposited to the edges affected by the swaps in order to determine the influence of the communication and bias other ants to explore possible improvements. This process is defined as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \frac{(\tau_{max} - \tau_{min})(1 - \omega)}{n}, \tag{5.4}$$

where $\omega \in (0, 1]$ is a parameter which indicates the degree of the communication influence (a good value is $\omega = 0.5$) and n is the problem size, e.g., number of cities for the TSP or number of customers for the VRP.

5.3 Experiments for the Static TSP and VRP

5.3.1 Experimental Setup

For the experiments in the TSP we have used the ACOTSP[1] implementation which contains all the algorithms described in Section 4.5. All the algorithms in the following experiments have been tested on a wide range of benchmark problem instances obtained from the TSPLIB[2]. For the experiments in the VRP we have used our own implementation based on the guidelines of the ACOTSP implementation. The algorithms used for the experiments are the ones described in Section 4.6, AS$_{rank}$-CVRP and MACS-VRP. Although MACS-VRP has been applied to the VRP with time-windows, we have modified the algorithm for the CVRP. The colony that optimizes the number of vehicles is not used, but the way ants construct solutions to minimize the routes of the vehicles is used. The algorithms have been tested on a wide range of benchmark problem instances obtained from the VRPLIB[3]. The VRP instances have two types of CVRP, with (C1-C5 and C11, C12) and without service time (C6-C10 and C13, C14). The CVRP with service time, apart from the capacity constraint Q, described in Chapter 4.6.2, has an additional constraint in which each vehicle must not exceed the service time L. The benchmark problem instances used for both TSP and VRP are described in Appendix A.

5.3.2 Parameter Settings

All the algorithms perform 5000 iterations and were executed for 30 independent runs for each problem instance in both TSP and VRP. Some of the parameters of

[1]Available at: http://www.aco-metaheuristic.org/aco-code by Thomas Stützle
[2]Available at: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
[3]Christofides, Mingozzi and Toth benchmark instances available at: http://neo.lcc.uma.es/radi-aeb/WebVRP/

the different ACO algorithms have been obtained from [69, p. 71] that suggest reasonable performance over a significant set of TSP instances and some others from our preliminary experiments. In the case of the VRP, some of the parameters suggested in [27, 28, 100] have been obtained and some others from our preliminary experiments again. Appendix B presents all the parameters used for all the algorithms in both TSP and VRP. However, due to the problem dependency, different parameter settings may result in better or worse performance.

5.3.3 Experimental Results for the Static TSP

5.3.3.1 Conventional ACO Algorithms Performance

In Tables 5.1 and 5.2, the mean results of the best solution and the total runtime averaged over 30 runs are presented for different conventional ACO algorithms for the TSP.

From Table 5.1, it can be observed that the best performing ACO algorithm regarding solution quality is the $\mathcal{MM}$AS. The main difference of $\mathcal{MM}$AS from the other ACO variations is that it keeps the pheromone trails to a certain range. In this way, it eliminates areas with a high intensity of pheromone trail values, and achieves more exploration, especially on the initial stages of the algorithm. Moreover, the population of ants is able to escape stagnation behaviour with the pheromone re-initialization mechanism. In fact, BWAS has a re-initialization mechanism but its performance is inferior to the $\mathcal{MM}$AS because the former does not keep the pheromone trail values to a certain range explicitly.

From Table 5.2, it is can be observed that the best performing ACO algorithm regarding the runtime is the ACS. The main difference of ACS to the other ACO variations is that the number of ants used is less than the other algorithms. In the

Table 5.1: Averaged results over 30 runs of the best solution for 5000 iterations of the algorithms. Bold values indicate the best results.

Instance	AS	EAS	AS$_{rank}$	$\mathcal{MM}$AS	ACS	BWAS
eil51	439.67	434.57	430.20	**426.44**	428.87	431.33
eil76	553.03	550.23	542.23	**538.72**	543.77	549.27
eil101	676.20	656.34	641.37	**632.28**	641.83	644.17
kroA100	22806.37	21852.90	21673.00	**21358.24**	21561.43	21706.00
kroA150	28259.33	27631.43	27428.13	**26976.56**	27161.93	27139.43
kroA200	31988.13	30586.27	29964.13	**29522.32**	29671.53	29799.50
lin318	464128.85	44145.20	43446.00	**42746.48**	43622.00	43317.03
pcb442	59309.13	54145.40	54270.53	**52425.76**	53064.00	53705.67
att532	31841.83	29875.50	29400.73	**28298.60**	29344.60	29032.67
rat783	10444.63	9530.00	9619.33	**9003.68**	9533.10	9279.17
pcb1173	70174.80	64714.37	68097.70	**63583.36**	64252.17	63646.70
pr2392	468461.17	465301.50	460700.33	**432841.30**	437465.77	440666.83

case more ants are used in the TSP, the solution quality is degraded. However, the ants in ACS update pheromone trails while they construct solutions, and they use a more aggressive decision rule as defined in Equation 4.13. ACS is more likely an extension of AS, rather than a variation, because it has more differences than similarities. For example, the pheromone evaporation is applied only to the trails of the best ant and not to all the pheromone trails as with the other algorithms. In fact, $\mathcal{MM}$AS has only the best ant to deposit pheromone as with ACS, but in the former evaporation is applied globally and it has additional processes to keep the pheromone trail values to a certain range and to detect stagnation behaviour in order to re-initialize the pheromone trails, which increase the total runtime. It is worth to mention that ACS also keeps the pheromone trails in a certain range, but implicitly.

Table 5.2: Averaged results over 30 runs of the total runtime (seconds) for 5000 iterations of the algorithms. Bold values indicate the best results.

Instance	AS	EAS	AS$_{rank}$	$\mathcal{MM}$AS	ACS	BWAS
eil51	3.26	2.10	2.02	2.04	**1.84**	1.88
eil76	5.98	3.88	5.15	**3.77**	4.79	4.98
eil101	8.95	6.08	7.80	**6.02**	6.42	7.88
kroA100	6.26	6.16	5.91	6.06	**3.96**	5.68
kroA150	15.91	12.15	14.50	12.02	**9.72**	14.78
kroA200	25.49	20.04	22.97	19.89	**12.79**	24.66
lin318	45.55	46.28	42.72	46.36	**13.11**	43.03
pcb442	84.52	84.24	79.75	85.52	**19.75**	82.40
att532	136.72	122.16	129.56	137.09	**36.23**	141.38
rat783	265.36	263.21	258.27	294.21	**38.30**	271.98
pcb1173	577.19	571.45	550.57	448.89	**58.97**	602.51
pr2392	2030.13	2047.50	1976.62	2111.17	**45.97**	2103.94

In general, all the variations and extensions of AS improve its performance regarding the solution quality significantly. Therefore, for the remaining experiments we consider $\mathcal{MM}$AS which is the best performing ACO regarding the solution quality. Experiments are performed by comparing $\mathcal{MM}$AS and $\mathcal{MM}$AS with the proposed scheme, denoted as $\mathcal{MM}$AS+DC, on different TSP benchmark problem instances.

5.3.3.2 Effect of Communication Range T_r

The communication range, i.e., T_r, is an important parameter for the proposed DC scheme in order to achieve a robust behaviour for the ACO algorithm. In Figure 5.2, different values of the T_r parameter are illustrated, which show the effect they have on the solution quality for different problem instances. Four different instances are

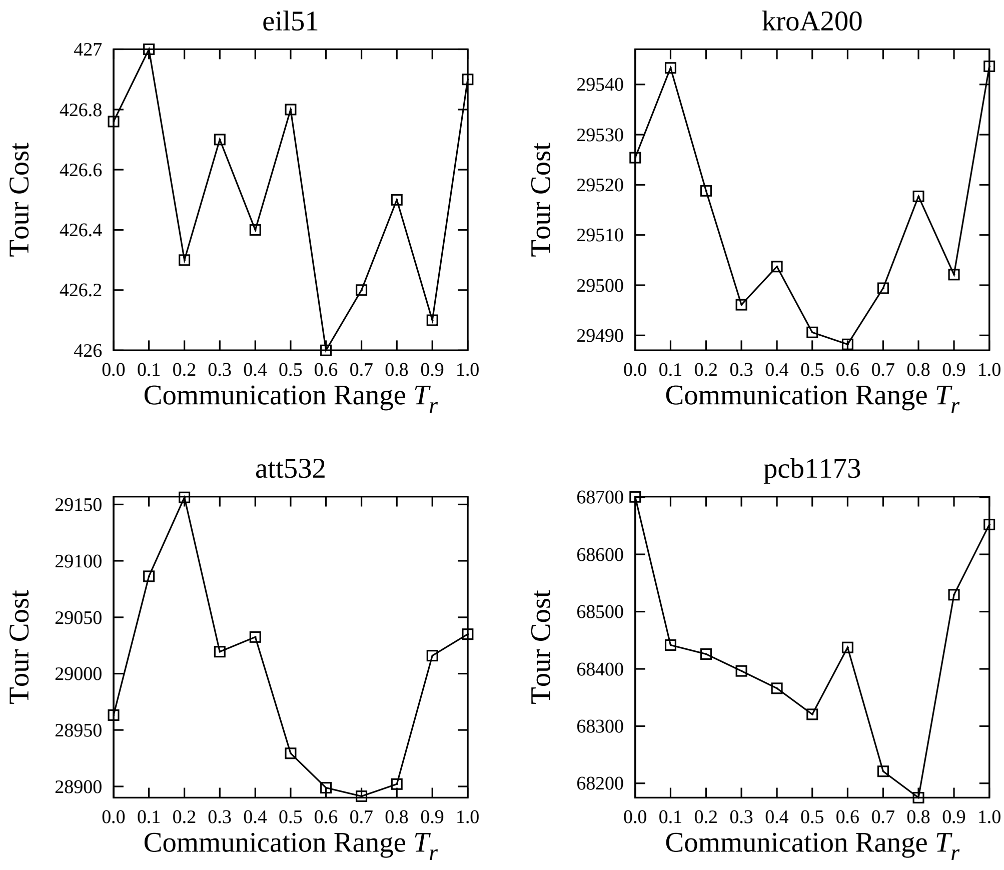


FIGURE 5.2: The effect of the communication range parameter T_r used in the DC scheme for different problem instances.

selected, i.e., eil51, kroA200, att532 and pcb1173, indicating small, to medium, and large problem instances, respectively. When the range parameter is 0.0, it means that the ants do not communicate with each other and, thus, we have a conventional $\mathcal{MM}$AS algorithm. On the other hand, if the range value is 1, it means that the ants are allowed to communicate with all other ants.

The parameter is problem dependent and hence depends on the size of the problem instance. As the problem size increases the communication range should increase. This can be observed from Figure 5.2 where a smaller range performs better on the smallest problem instance, i.e., eil51, and a larger range performs better on the largest problem instance, i.e., pcb1173.

Moreover, it can be observed that when $T_r = 0.1$ on some problem instances, i.e., eil51, kroA200, and att532, the performance of the algorithm is worse. This is probably due to the fact that more similar ants communicate, because of the limited communication range, and may get stuck in stagnation behaviour faster.

In general, a good range is $0.6 \leq T_r \leq 0.8$ which means that it would be good for the ants to communicate with dissimilar ants but to generate a reasonable size of a neighbourhood. A larger value of the range may lead the population into a higher level of diversity, which may result in randomization, whereas a smaller value may be ineffective since the ants selected have more chances to be similar. An appropriate communication range influences ants to concentrate on the paths found from the ACO searching, and not disturb the optimization process. Hence, in the experiments of the next subsection we have set $T_r = 0.6$ for eil51, eil76, eil101, kroA100, kroA150 and kroA200, $T_r = 0.8$ for pcb1173 and pr2392, and $T_r = 0.7$ for the remaining problem instances.

5.3.3.3 $\mathcal{MM}$AS vs $\mathcal{MM}$AS+DC Performance

In Table 5.3, the mean and standard deviations results of $\mathcal{MM}$AS and $\mathcal{MM}$AS+DC are presented for the different TSPs. The corresponding statistical results of comparing the algorithms by a non-parametric statistical test, i.e., Wilcoxon rank-sum test, at the 0.05 level of significance are also shown in the last column.

In general, it can be observed that the solution quality of $\mathcal{MM}$AS is significantly improved for most problem instances when the DC scheme is used. More precisely, on small problem instances, the solution quality is slightly improved whereas on larger problem instances the improvement is significantly better. This is due to the fact that for small problem instances $\mathcal{MM}$AS will converge to a single solution quickly. Therefore, neighbourhoods which consist of dissimilar ants cannot be easily

Table 5.3: Averaged results over 30 runs of the best solution for 5000 iterations of the algorithms with the corresponding standard deviations and statistical tests. "$-$" or "$+$" indicates that the first or the second algorithm is significantly better, respectively, and "$\sim$" indicates no statistical significance.

Instance	$\mathcal{MM}$AS		$\mathcal{MM}$AS+DC		Wilcoxon
	Mean	Std Dev	Mean	Std Dev	
eil51	426.64	0.76	426.36	0.49	$\sim$
eil76	538.72	1.17	538.56	0.82	$\sim$
eil101	632.28	2.85	631.36	3.07	$\sim$
kroA100	21358.24	52.99	21322.76	49.96	$+$
kroA150	26976.56	94.61	26910.68	88.36	$+$
kroA200	29522.32	87.21	29491.36	44.52	$\sim$
lin318	42746.48	221.74	42618.16	215.47	$+$
pcb442	52425.76	610.58	52036.80	592.49	$+$
att532	28298.60	178.03	28104.60	255.85	$+$
rat783	9003.68	51.90	8988.00	38.84	$\sim$
pcb1173	63583.36	687.46	63143.48	749.26	$+$
pr2392	431250.60	4287.98	428917.20	3267.67	$+$

generated in order for the DC scheme to work properly. The DC scheme tries to become effective using random swaps, but it may take some time due to the lower probability to avoid randomization. However, it may also become effective when $\mathcal{MM}$AS re-initializes its pheromone trails.

On large problem instances, the conventional $\mathcal{MM}$AS has more chances to get trapped on a local optimum since the search space is larger. Large problem instances require more exploration but $\mathcal{MM}$AS loses its exploration ability quickly because of the stagnation behaviour. Therefore, the DC scheme is effective on such cases since different neighbourhoods of dissimilar ants are generated and provide guided exploration. This can be observed from Table 5.3 in which $\mathcal{MM}$AS+DC is significantly better on most large problem instances, e.g., pcb1173 and pr2392.

5.3.4 Experimental Results for the Static VRP

5.3.4.1 Conventional ACO Algorithms Performance

In Table 5.4 and Figures 5.3 and 5.4, the mean results of the best solution, and the behaviour of the algorithms on each iteration, over 30 runs are presented, respectively, for conventional ACO algorithms on different VRP instances.

From Table 5.4, it can be observed that AS$_{rank}$-CVRP outperforms MACS-VRP in most random VRP instances, whereas it is outperformed by MACS-VRP in most clustered VRP instances.

From Figures 5.3 and 5.4 it can be observed that the MACS-VRP has a faster convergence than AS$_{rank}$-CVRP in almost all problem instances. This is due to the different decision rule used by the MACS-VRP, i.e., pseudorandom decision rule, which is more aggressive than the one used in AS$_{rank}$-CVRP, and achieves more exploitation. More precisely, using the traditional decision rule, the ants select the next object to move probabilistically using the roulette wheel method (a popular technique proposed on EAs), whereas with the pseudorandom decision rule, the ants usually select the object with the highest probability.

AS$_{rank}$-CVRP performs better than MACS-VRP on 7 problems instances, i.e., C1, C2, C3, C5, C6, C7 and C12, whereas MACS-VRP performs better than AS$_{rank}$-CVRP on the remaining problem instances. In general, AS$_{rank}$-CVRP is comparable with MACS-VRP in different test cases. The former algorithm has been proposed specifically for the CVRP, whereas the latter has been modified for the CVRP. Therefore, for the remaining experiments we consider AS$_{rank}$-CVRP by comparing

TABLE 5.4: The mean results of the best solution averaged over 30 runs for the CVRP with and without service time constraint. Bold values indicate the best results.

Instance	AS_{rank}-CVRP	MACS-VRP
Random Problems		
C1	**551.16**	564.76
C2	**898.16**	922.56
C3	**903.73**	907.86
C4	1166.30	**1160.6**
C5	**1442.36**	1456.03
C6	**593.30**	606.33
C7	**982.63**	998.5
C8	970.16	**962.43**
C9	1294.53	**1273.53**
C10	1577.96	**1553.9**
Clustered Problems		
C11	1171.56	**1116.33**
C12	**962.50**	964.93
C13	1687.20	**1548.66**
C14	1012.70	**967.10**

the conventional one with the AS_{rank}-CVRP with the proposed DC scheme, denoted as AS_{rank}-CVRP+DC.

5.3.4.2 AS_{rank}-CVRP vs AS_{rank}-CVRP+DC Performance

In Table 5.5, the mean results of AS_{rank}-CVRP and AS_{rank}-CVRP+DC are presented for the different CVRPs. The corresponding statistical results of comparing the algorithms by a non-parametric statistical test, i.e., Wilcoxon rank sum test, at

TABLE 5.5: The mean results of the best solution with the number of vehicles used averaged over 30 runs for the CVRP with and without service time constraint. "−" or "+" indicates that the first or the second algorithm is significantly better, respectively, and "∼" indicates no statistical significance.

Instance	AS$_{rank}$-CVRP		AS$_{rank}$-CVRP+DC		Wilcoxon
	Best route	Vehicles used	Best route	Vehicles used	
			Random Problems		
C1	551.16	5.13	545.73	5.06	+
C2	898.16	10.73	890.23	10.46	+
C3	903.73	8	894.00	8	+
C4	1166.30	12	1150.43	12	+
C5	1442.36	17	1435.1	17	∼
C6	593.30	6	588.73	6.06	∼
C7	982.63	12.03	971.16	12	+
C8	970.16	9	957.43	9	+
C9	1294.53	14.5	1277.73	14.56	+
C10	1577.96	19	1558.60	19	+
			Clustered Problems		
C11	1171.56	9.93	1123.16	9.73	+
C12	962.50	10	943.90	10	+
C13	1687.20	12.8	1646.03	12.86	+
C14	1012.70	11	983.56	11	+

the 0.05 level of significance are also shown in the last column. Note that the communication range of AS$_{rank}$-CVRP+DC was set to $T_r = 0.8$ for all problem instances, since it has been found to be a good value for the TSP previously.

In both CVRP variations, the AS$_{rank}$-CVRP+DC algorithm outperforms the conventional AS$_{rank}$-CVRP algorithm on almost all the problem instances, which can

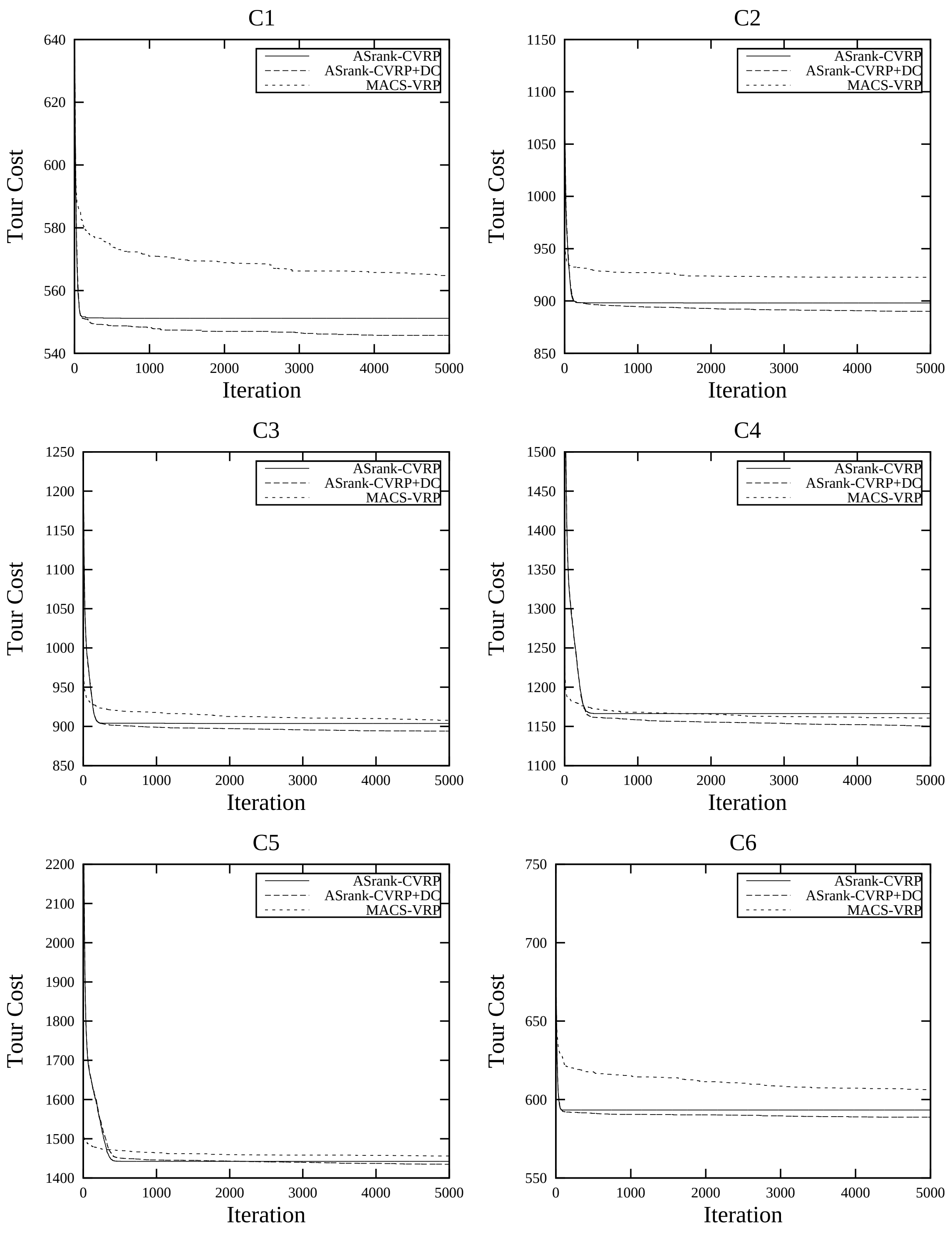


Figure 5.3: Behaviour of ACO algorithms on the random VRP instances, averaged over 30 runs.

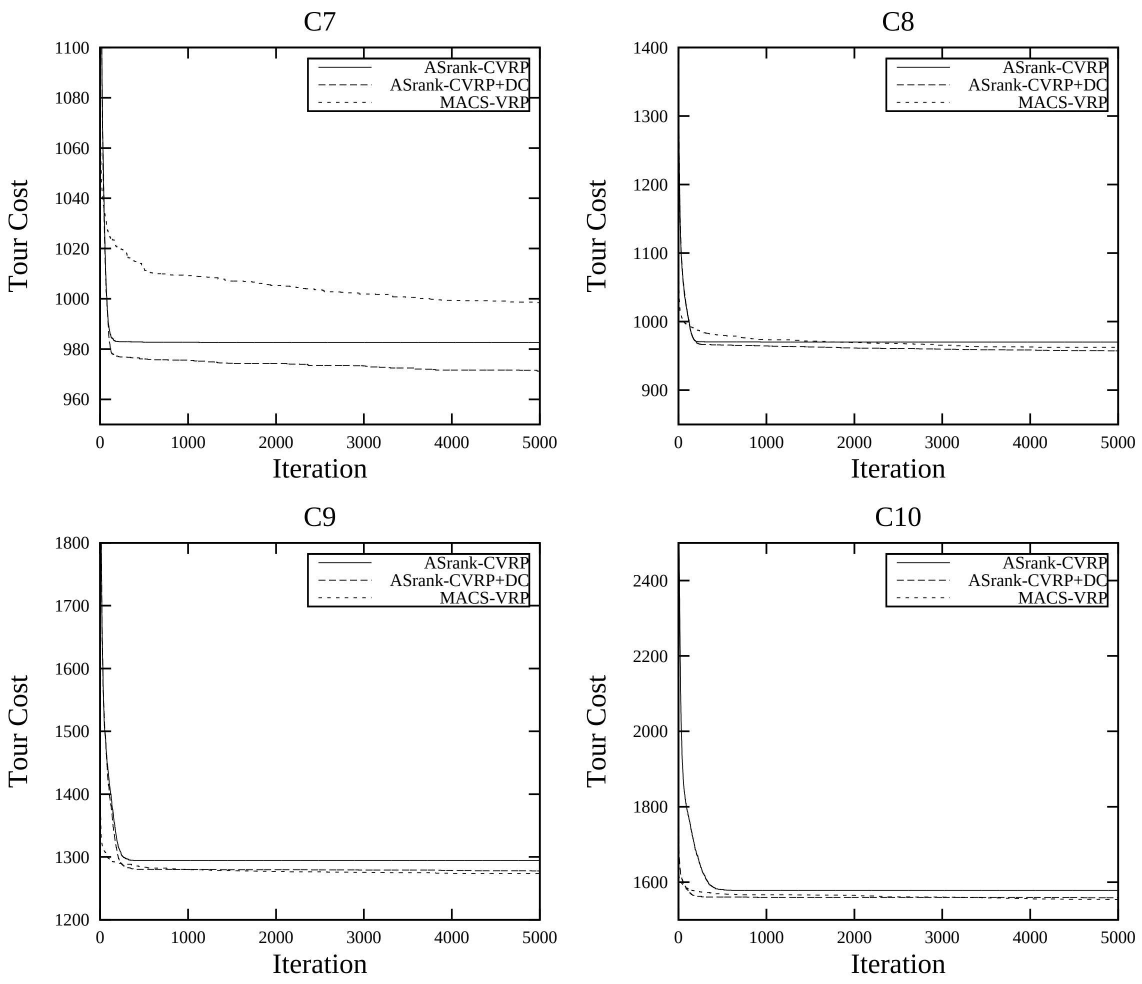


FIGURE 5.3: (continued)

be observed from Table 5.5, where the solution quality of AS$_{rank}$-CVRP+DC is significantly different. This is because the conventional AS$_{rank}$-CVRP algorithm has more chances to get trapped on a local optimum due to the stagnation behaviour.

On the other hand, it can be observed from Figures 5.3 and 5.4, that AS$_{rank}$-CVRP gets trapped in a local optimum in almost all problem instances. The proposed AS$_{rank}$-CVRP+DC algorithm increases the exploration ability and escapes from possible local optima using the swaps. Moreover, the extra local pheromone update of the DC scheme to the possible improvements found from the swaps may attract ants to explore promising areas in the search space.

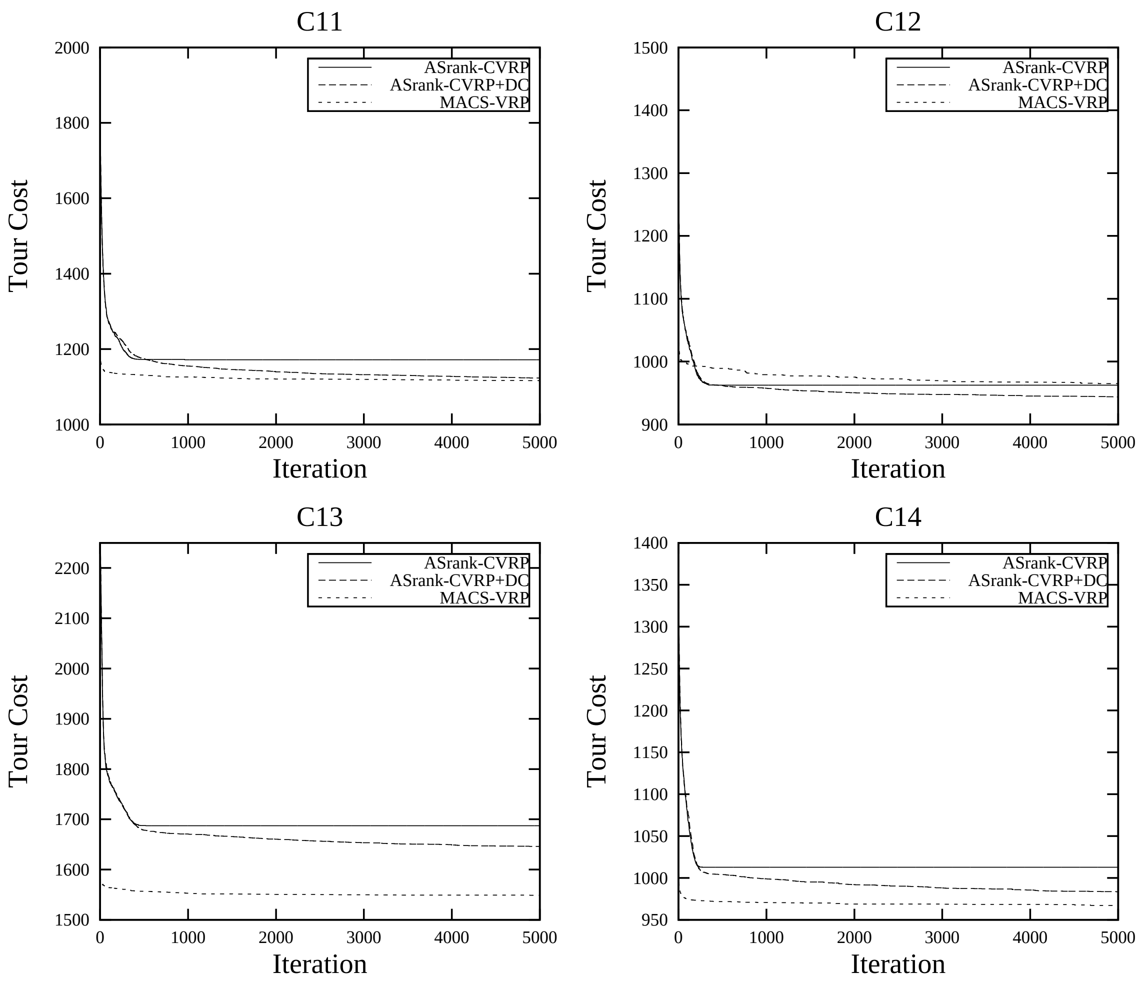


FIGURE 5.4: Behaviour of ACO algorithms on the clustered VRP instances, averaged over 30 runs.

5.4 Summary

In this chapter we have presented a scheme that improves the conventional ACO performance in static environments. In conventional ACO algorithms, ants communicate indirectly via their pheromone trails. In the proposed scheme, ants communicate both indirectly and directly. The proposed DC scheme enables ants to exchange information after they construct their solutions. Ants communicate directly with other ants within a predefined communication range. The scheme is based on a combination of adaptive and random swaps, in which the former promotes the exchange of information between ants, whereas the latter promotes exploration once the algorithm reaches the stagnation behaviour.

In the experiments for the TSP we have compared several ACO variations, explained in Chapter 4 before, in which $\mathcal{MMAS}$ performs better regarding the solution quality than its competitors, whereas ACS performs better regarding computation time than its competitors. Furthermore, the proposed DC scheme was integrated to the $\mathcal{MMAS}$ algorithm and showed that often improves the performance of a conventional $\mathcal{MMAS}$ significantly, if an appropriate communication range is chosen.

In the experiments for the VRP we have compared two ACO variations, explained in Chapter 4 before, in which AS_{rank}-CVRP outperforms MACS-VRP on some problem instances, whereas MACS-VRP outperforms AS_{rank}-CVRP on the remaining problem instances. Furthermore, the proposed DC scheme was integrated to the AS_{rank}-CVRP and showed that in almost all test problem instances improves the performance of a conventional AS_{rank}-CVRP significantly.

In general, we have showed that the DC scheme improves the performance of ACO in the TSP and the CVRP, since it helps the population to escape from possible local optima.

Chapter 6

ACO for Dynamic Combinatorial Optimization

6.1 Dynamic Optimization Problems (DOPs)

Traditionally, researchers are focused to SOPs, in which a problem remains unchanged during the execution of an algorithm. However, in many real-world applications we have to deal with DOPs. Formally, a problem instance of a DOP can be defined as:

$$\Pi = (X(t), \Omega(t), f(t))_{t \in S}, \tag{6.1}$$

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

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

The environmental changes are classified into two types: dimensional and non-dimensional changes, where both of them cause the optimum to change on every environmental change. Dimensional changes correspond to adding or removing variables from the problem. Such environmental changes affect the representation of the solutions and alter a feasible solution to an infeasible one. A repair operator may address this problem, but requires prior knowledge of the problem and the detection of the dynamic changes. Non-dimensional changes correspond to the change of the variables of the problem. Such environmental changes do not affect the representation of the solutions, and, thus, are more straightforward to address.

In Chapter 3, we have discussed the challenges to algorithms in solving $\mathcal{NP}-$complete COPs with static environment. Addressing $\mathcal{NP}-$complete COPs with dynamic environment, is even more challenging for algorithms because the objective is not only to locate the global optimum efficiently, but to also track it during the environmental changes [24].

A straightforward method to address DOPs is to consider every dynamic change as the arrival of a new problem instance that needs to be solved from scratch. However, such a method: (1) requires substantial computational effort; (2) may take long to re-optimize; and (3) destroys all the domain-specific information. Furthermore, it requires the detection of changes which is often difficult to achieve in DOPs.

Nature-inspired metaheuristics are used to address DOPs, since they are inspired from nature which is a continuous adaptation process [137]. Such methods are able to transfer knowledge from previous environments since they are iterative. In this way, it may be possible to speed up the re-optimization time with the use of the adaptation capabilities of nature-inspired metaheuristics, e.g., EAs and ACO algorithms, assuming that the changing environments have similarities. Otherwise, a global restart of the algorithm may be the better option.

However, even if knowledge is transferred from previous environments the algorithms need to be flexible enough to accept it and adapt well to the new environment. The challenge for EAs and ACO algorithms lies in that they loose their adaptation capabilities early due to the premature convergence or stagnation behaviour. This is because traditional EAs and ACO algorithms have been designed to tackle SOPs and the diversity of solutions is decreased as the algorithms progress to optimize. Several strategies have been integrated with the algorithms to vary the balance between exploration and exploitation in order to adapt well in dynamic environments.

6.2 Optimization Methods for DOPs

6.2.1 Evolutionary Computation

Mainly, the research to address DOPs is focused on EAs. EAs are able to transfer knowledge since the information of previous environment is stored in the population of individuals of the previous iterations. However, the individuals in the old environment may not be feasible for the new one but, if they are re-evaluated or repaired, they may transfer valuable information. Several surveys and books are available for EDO [21, 137, 193, 200, 264, 279, 281].

An EA achieves exploitation via selection and crossover operators, whereas exploration is achieved via the mutation operator. Usually, the mutation operator has a small probability in order for the algorithm to converge to an optimum. In DOPs, once the population converges to an optimum, then it is difficult for the population to track the moving optimum after a change.

Many strategies have been proposed and integrated with EAs to improve the re-optimization time and maintain a high quality of the output efficiently, simultaneously. The main contributions of these strategies are categorized as follows.

6.2.1.1 Increasing Diversity after a Change

The methods in this category generate diversity after a dynamic change occurs. Cobb [41] introduced hyper-mutation, in which the probability of mutation is increased drastically, after an environmental change, for a number of iterations to generate diversity. Vavak et al. [256–258] introduced the variable local search (VLS) in which the mutation probability is increased gradually.

Hyper-mutation and VLS require the detection of a change and may not be compatible in DOPs where the changes are undetectable. Moreover, the diversity is increased randomly and valuable information found in the previous iterations of the algorithm may be destroyed.

6.2.1.2 Maintaining Diversity during the Execution

The methods in this category maintain diversity throughout the run, not only after a change occurs. Grefenestette [107] introduced random immigrants, in which randomly generated individuals replace a small portion of the population (usually the worst) in every algorithmic iteration. However, this immigrants scheme may disturb the optimization process since diversity is generated randomly.

Yang [274] introduced the elitism-based immigrants, in which immigrants are generated using the best individual of the previous environment as the base. In this way, diversity is guided but this immigrants scheme may work only when the changing environments are similar. A hybrid immigrants scheme was introduced later on that combines random, dualism and elitism-based immigrants [280].

Yu et al. [288] introduced environmental-information based immigrants to address slightly changing environments. The knowledge transferred from this immigrants scheme is based on the whole population of the previous environment and not in one individual as with the elitism-based immigrants.

It has been observed that individual-information immigrants have better *performance* than environmental-information immigrants, whereas the latter one has better *robustness* than the former one. Individual- and environmental-information based immigrants have been hybridized to combine the merits of both schemes [290]. From the experiments it has been observed that: (1) robustness and performance cannot be optimized simultaneously; and (2) the interaction between the two types of immigrants does strike a balance between robustness and performance.

Apart from immigrants schemes, co-evolutionary techniques, e.g., fitness sharing and crowding [36, 172], help to enhance population diversity. Co-evolution has significantly improved evolution in optimization problems [127]. Fitness sharing modifies the objective value of individuals to create niches. Individuals that belong to the same niche share their fitness. Crowding is a replacement strategy in which similar individuals are replaced. Both techniques delay premature convergence, and, thus they maintain diversity.

Immigrants schemes and co-evolutionary techniques slow down the optimization process, since they enhance exploration in every iteration. Hence, in dynamic environments that change quickly, they may not be effective.

6.2.1.3 Memory-based schemes

The methods in this category involve memory to the algorithms that store solutions or knowledge from previous environments that can be reused in the future. Memory-based schemes are further divided into two categories: *implicit* and *explicit* memory.

The former type of memory refers to the integration of EAs with redundant representation. The most popular examples of implicit memory are the diploid EAs [109, 273], where each locus (position) in an individual has two alleles (values).

The latter type of memory refers to the integration of EAs with a memory component to store information [275, 277, 291]. Memory-based EAs differ in the way information is stored and retrieved from the memory, and when and how memory is updated.

Memory-based schemes are useful on dynamic environments when previous environments reappear; otherwise, they might not be effective [20].

6.2.1.4 Multi-population approaches

The methods in this category maintain several sub-populations, in parallel, where each one tracks a different area in the search space. Multi-population EAs differ in the way the sub-populations are used.

In self-organizing scouts [23], a main population is used to explore the search space and locate an optimum, whereas a sub-population is created whenever a new optimum is found, and is used to track the new optimum. In shifting balance GA [265], a main population is used to track any new optimum found, whereas several sub-populations explore the search space. In multinational GA [253], several sub-populations have the ability to both explore and track a location in the search space.

Multi-population approaches have the ability to generate and maintain diversity. However, a large number of sub-populations may require extensive computation time. Moreover, it may slow down the optimization process due to the large number of parallel locations in the search space that the populations track.

6.2.1.5 Hybrid and memetic algorithms

These methods are combinations of the above approaches. Every approach, from the above categories, may perform better on different dynamic environments with different properties. For example, memory-based approaches have been integrated with immigrants schemes [235, 275]. The former approaches promotes guidance, and the latter approaches generate diversity.

Another type of hybrid algorithms, called memetic algorithms, have been applied successfully on DOPs. A memetic algorithm is an EA with an LS operator. Usually, a memetic algorithm provides strong exploitation and may not be suitable for DOPs. However, immigrants schemes have been integrated with a memetic algorithm to balance exploitation and exploration with promising results [261]. Other memetic algorithms have been successfully applied in different dynamic environments [80, 81, 131].

6.2.2 Other Metaheuristics

On the contrast, the research to address DOPs using other metaheuristics is weak compared with EAs. A parallel implementation of TS has been applied to the real-time vehicle routing and dispatching with promising results [106]. SA has been applied to the dynamic load balancing problem to obtain a satisfactory high-performance scheduling [204]. EDAs enhanced with memory have been applied on different binary-encoded dynamic COPs [283]. The ACO metaheuristic in dynamic environments will be discussed in great detail in Section 6.5.

6.3 Detection of Dynamic Changes

Many strategies used to enhance the performance of metaheuristics in DOPs, e.g., hypermutation described above, take explicit action when a change occurs in dynamic environments and require the detection of changes. It can be assumed that the environment makes the changes known to the algorithm, but this may not be the case in real-world applications. Therefore, two main strategies are integrated with the algorithms to enable the detection of dynamic changes: (1) detect a change by re-evaluating detectors (special locations in the search space where the objectives are re-evaluated on every iteration to detect changes); and (2) detect a change by considering the algorithm's behaviour.

The former method refers to the regular re-evaluation of detectors from the algorithm to detect changes of their value or their feasibility. The detectors can be part of the actual population, e.g., the best solutions, [130, 157] or a separate sub-population [275, 300]. Since the detectors require function evaluations, usually a small number of detectors is used. However, sometimes the detectors may not be able to detect the change since a dynamic change may not affect the location of the detectors. Several studies have been performed for the optimal number of detectors and the location [193, 222], e.g., using a memory where the best solutions of the previous environment are used as detectors [278]. In general, this method guarantees to detect a dynamic change and it is commonly used in EAs for dynamic environments [278]. A drawback of this method lies in that when noise (a very small environmental change) exists in the function evaluation, for this case, noise may mislead the detectors to detect a dynamic change. Usually for this case, noise exists in every algorithmic iteration, and, thus the detectors will always detect a dynamic change [135].

The latter method refers to the monitoring of the algorithm behaviour. For example, in [41] a dynamic change is detected by monitoring a drop on the value of the best solution over a number of iterations, whereas in [193] the diversity is monitored. This method does not require any extra fitness evaluations as the former method, but there is no guarantee that a dynamic change will be detected [135, 222].

6.4 Performance Measurements

In dynamic optimization there is no agreed unique measurement to evaluate the performance of algorithms. To simply consider the best solution found may not be sufficient because the optimum changes. There are three categories of performance measurements: (1) optimum-based performance measurements; (2) behaviour-based performance measurements; and (3) average-based performance measurements.

The optimum-based performance measurements measure the best an algorithm can perform. For example, how close to the optimum an algorithm can perform on every dynamic change. The behaviour-based performance measurements measure different behaviours of the algorithms. For example, how different are the solutions in the population or how well an algorithm reacts after a dynamic change. In the average-based performance measurements the best individuals are not so important as in the optimum-based performance measurements. This is because it measures the population as a whole, i.e., the average fitness values of the population. Some researchers often adopt average-based measures since they use (population-based) algorithms as a model of evolutionary systems [214]. The most popular and common measurements in DOPs are categorized and discussed below.

6.4.1 Optimum-based Performance Measurements

Best-performance. Best-performance (BP) is one of the most commonly used measurements to compare algorithms in DOPs. Different names have been given: best-of-generation [282], best-fitness [103], best-objective-value [6] and collective mean fitness [192]. Generally, BP is the average of the best values on each iteration for several runs on the same problem with the same dynamic changes. The best-performance is defined as follows:

$$\bar{P}_{BP} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N} F_{BP_{ij}}\right), \tag{6.2}$$

where G is the number if iterations (or generations), N is the number of runs, and $F_{BP_{ij}}$ is the best value in iteration i of run j.

The advantage of BP is that it gives an overall view of the performance of the algorithms in dynamic environments, and it is easier to quantitatively compare them. The disadvantages is that the optimum is not considered, and, thus, the values are not normalized. Therefore, in case an algorithm performs well after a change, and has a very poor fitness after a change on a specific period, the final value may be biased and might not reflect correctly its overall performance.

Modified offline-performance. Offline-performance was initially proposed for stationary environments and later on it was modified to evaluate dynamic environments [21, 137]. The modified offline-performance (MOP) is the average over the best value found since the last environmental change and is defined as follows:

$$P_{MOP} = \frac{1}{G}\sum_{i=1}^{G} F_i^*, \tag{6.3}$$

where F_i^* is the best fitness value since the last environmental change in iteration i.

The advantages and disadvantages of MOP are similar with the BP above. However, it has an additional disadvantage because it requires that the period of a dynamic change is known.

Modified offline-error. In best-performance and modified offline-performance measurements the optimum is not known. The modified offline-error (MOE) [22] is the average over the best error value found since the last environmental change and it is defined as follows:

$$P_{MOE} = \frac{1}{G}\sum_{i=1}^{G} E_i^*, \tag{6.4}$$

where E_i^* is the best error value since the last environmental change in iteration i. The error is a metric that defines the difference between the best fitness value and the known global optimum value, where a value closer to 0 means better performance.

MOE has the same advantages and disadvantages as MOP. An additional disadvantage is that it requires that the moving optimum is known for every period.

Best-error. Similarly with the modified offline-error measurement, the best-error (BE) measures how close to the optimum an algorithm can perform [251]. However, it considers the average error achieved just before an environmental change, and not the error of the best fitness value since the last environmental change as in Equation 6.4. The BE is defined as follows:

$$P_{BE} = \frac{1}{M}\sum_{i=1}^{M} E_i, \tag{6.5}$$

where M is the number of environmental changes and E_i is the best error value before the i-th iteration.

The advantage of BE is that it can give a clear comparison of the final outcome of the algorithms. However, it has many disadvantages such as: (1) the moving optimum has to be known on each period in addition with the time period of the dynamic change; (2) the values of the errors are not normalized and may bias the final values; and (3) the recovery time of the algorithm until it reaches the best fitness value before a change is not considered. Therefore, an algorithm that recovers very slow and reaches the same value as another algorithm that recovers much faster, will have the same BE as that algorithm.

Relative-error. The relative-error (RE) measures the optimization accuracy assuming that the best and the worst value in the search space are known [263]. RE in iteration i can be defined as follows:

$$P_{RE_i} = \frac{F_{BP_i} - F_{worst_i}}{F_{best_i} - F_{worst_i}}, \tag{6.6}$$

where F_{BP_i} is the best value of the algorithm, F_{best_i} is the best value in the search space, and F_{worst_i} is the worst value in the search space at iteration i, respectively. The value of P_{RE_i} ranges between 0 and 1, where 1 is the best possible value.

The advantages of RE are the same as those of BP, MOP and MOE. However, the values of RE are normalized and they are less biased in the periods where the fitness is significantly different than in other periods. The disadvantage is that it requires that the best and the worst values in the search space for every period are known.

6.4.2 Behaviour-based Performance Measurements

Total-diversity. The total-diversity (TD) measurement tends to measure how different the solutions in the population are. Depending on the encoding of the problem there are different metrics to define the diversity. Usually, on binary-encoded problems the hamming-distance is used [202, 278, 290], whereas on permutation-encoded problems a metric based on the similarities of the objects is used [3]. The TD in iteration i can be defined as follows:

$$\bar{T}_{DIV} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N} Div_{ij}\right), \tag{6.7}$$

where in case a binary-encoded representation is used, Div_{ij} is defined as:

$$Div_{ij} = \frac{1}{l\mu(\mu-1)}\sum_{p=1}^{\mu}\sum_{q\neq p}^{\mu} HD(p,q), \tag{6.8}$$

where l is the encoding length, μ is the population size, and $HD(p,q)$ is the hamming distance between solutions p and q. In case a permutation-encoded representation is used, Div_{ij} is defined as:

$$Div_{ij} = \frac{1}{\mu(\mu-1)}\sum_{p=1}^{\mu}\sum_{q\neq p}^{\mu} S(p,q), \tag{6.9}$$

where $S(p,q)$ is the similarity metric based on the common edges between solution p and q defined in Equation 5.2. For both metrics in Equations 6.8 and 6.9 a value closer to 0 means that the solutions in the population are similar.

TD can provide important information for the algorithm in order to tune the balance between exploration and exploitation. For example, if an algorithm has high diversity, it means that the algorithm generates too much exploration, which may lead to randomization.

Stability. The stability measurement measures the increase of the fitness after a dynamic change occurs [263]. An algorithm is called "stable" when its optimization accuracy is not affected significantly when a dynamic change occurs. Therefore, using the optimization accuracy, defined in Equation 6.6, the stability of an adaptive algorithm in iteration i is defined as follows:

$$S_i = \max\{0, P_{RE_{i-1}} - P_{RE_i}\}, \tag{6.10}$$

where $P_{RE_{i-1}}$ and P_{RE_i} are the optimization accuracy values just before and when a dynamic change occurs, respectively. The value of S_i ranges between 0 and 1, where 0 is the best possible value.

Similarly with TD, stability provides important information regarding the adaptation capabilities of the algorithm. However, it inherits the disadvantages of RE above since is based on that.

Best-robustness. The best-robustness (BR) measurement measures the response of the algorithm when a dynamic change occurs, and it is similar with stability [136, 290]. The BR for iteration i can be defined as follows:

$$\bar{R}_{BR_i} = \frac{1}{N}\sum_{j=1}^{N} BR_{ij} \tag{6.11}$$

where BR_{ij} is the best-robustness of iteration i of run j which is defined as follows:

$$BR_{ij} = \begin{cases} 1, & \text{if } \frac{F_{BP_{i-1j}}}{F_{BP_{ij}}} > 1, \\ \frac{F_{BP_{i-1j}}}{F_{BP_{ij}}}, & \text{otherwise}, \end{cases} \tag{6.12}$$

where $F_{BP_{i-1j}}$ and $F_{BP_{ij}}$ are the best values found by the adaptive algorithm, before and when an environmental change occurs, respectively. A higher value implies a better robustness.

BR can provide similar information with stability. However, it inherits the disadvantages of BP above because it is based on that. Recently, a new perspective on DOPs has been established, known as robust optimization over time (ROOT), where the target is to find the sequence of solutions which are robust over time [287]. More precisely, a solution is robust over time when its quality is acceptable to the environmental changes during a given time interval.

Reactivity. The reactivity measurement measures the time an adaptive algorithm requires to recover after a dynamic change [263]. Using the optimization accuracy defined in Equation 6.6, reactivity is defined as follows:

$$R_{i,\epsilon} = \min\{i' - i | i < i' \leq G, i' \in \mathbb{N}, \frac{P_{RE'_i}}{P_{RE_i}} \geq (1-\epsilon)\} \cup \{G - i\}, \tag{6.13}$$

where ϵ is a constant that defines the time window and G is defined as in Equation 6.2.

The advantages and disadvantages of the reactivity measurements are the same as those of the stability measurement.

6.4.3 Average-based Performance Measurements

Average-performance The average-performance (AP) measurement [214, 290] is the corresponding average-based performance of the best-performance defined in Equation 6.2. AP is defined as follows:

$$\bar{P}_{AP} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N} F_{AP_{ij}}\right), \tag{6.14}$$

where $F_{AP_{ij}}$ is the average of the fitness values of the population on iteration i of run j, and it is defined as:

$$F_{AP_{ij}} = \frac{1}{\mu}\sum_{k=1}^{\mu} F_k, \tag{6.15}$$

where F_k is the fitness value of the k-th individual.

Average-robustness The average-robustness (AR) measurement [214, 290] is the corresponding average-based performance of the best-robustness defined in Equation 6.12. Therefore, based on the AP measurement defined in Equation 6.15, the

AR for iteration i is defined as follows:

$$\bar{R}_{AR_i} = \frac{1}{N}\sum_{j=1}^{N} AR_{ij} \tag{6.16}$$

where AR_{ij} is the average-robustness of iteration i of run j which is defined as follows:

$$AR_{ij} = \begin{cases} 1, & \text{if } \frac{F_{AP_{i-1j}}}{F_{AP_{ij}}} > 1, \\ \frac{F_{AP_{i-1j}}}{F_{AP_{ij}}}, & \text{otherwise.} \end{cases} \tag{6.17}$$

6.5 Addressing Dynamic Environments with ACO

Differently from EAs, the research of ACO for DOPs is still infant. ACO algorithms are able to use knowledge from previous environments using the pheromone trails generated in the previous iterations. They also have adaptation capabilities due to pheromone evaporation [17]. For example, when a dynamic change occurs, evaporation will eliminate the pheromone trails concentrated to the old optimum. In this way, the population of ants will be flexible enough to track the moving optimum.

ACO achieves exploitation due to the consideration of heuristic information of the problem, and exploration due to the use of the pheromone mechanism. ACO algorithms are robust since they can accept any knowledge transferred via their pheromone trails: when the information is useful then it is considered; otherwise, it is destroyed after a few iterations, depending on the evaporation rate.

However, the time required to adapt to the new environment may depend on the problem size and the degree of change. When the environmental change is severe then it will take an ACO algorithm longer to eliminate unused pheromone trails. On the other hand, pheromone evaporation may destroy information that can be used on further environments, since a bad solution in the current environment may be good in the next environments [4, 82]. Several strategies have been proposed to enhance the performance of ACO in DOPs and they are categorized as follows.

6.5.1 Pheromone modification after a change

A global restart of an ACO algorithm is performed with the re-initialization of the pheromone trails. This mechanism was first proposed in $\mathcal{MM}$AS [244], and later on adapted in ACS [101] and BWAS [42] algorithms, to avoid the stagnation behaviour in static environments. Guntsch et. al [111, 114] used pheromone re-initialization and proposed the $Restart - strategy$ for DOPs. When a dynamic change occurs, all the pheromone trails are re-initialized with an equal amount.

However, this strategy destroys all the previous information and may slow down the re-optimization process. The same authors proposed local restart strategies that take into account where the change of the problem occurs [111, 114]. Two strategies have been proposed called $\tau - strategy$ and $\eta - strategy$, in which the former one modifies pheromone in the affected areas considering the values of the existing pheromone, whereas the latter one considers heuristic information. From their results, the local restart strategies performed better than the global one [111].

Eyckelhof and Snoek [82] proposed a “shaking” operator that is applied after a dynamic change is detected. The purpose of the operator is to smooth all the pheromone trails accordingly after pheromone evaporation. Areas with high intensity of pheromone may take some time to be eliminated by pheromone evaporation. Therefore, the shaking operator aims to eliminate them faster.

The methods in this category require the detection of dynamic changes. Moreover, $\tau - strategy$ and $\eta - strategy$ require information regarding the location of dynamic changes. Such assumptions may not be available in many DOPs. The

Algorithm 11 Ant Colony Optimization for DOPs

```
1: $t \leftarrow 0$
2: $P(0) \leftarrow$ InitializePopulation($\mu$)
3: InitializePheromoneTrails($\tau_0$)
4: $x^{bs} \leftarrow$ *empty solution*
5: **while** (termination condition *not* met) **do**
6:     $P(t) \leftarrow$ ConstructAntSolutions
7:     PheromoneEvaporation($\rho$)
8:     DepositPheromone($P(t)$)
9:     $x^{ib} \leftarrow$ FindBest($P(t)$)
10:    **if** ($f(x^{ib}) < f(x^{bs})$) **then**
11:        $x^{bs} \leftarrow x^{ib}$
12:    **end if**
13:    **if** (environmental change is *detected*) **then**
14:        *DaemonActions*
15:    **end if**
16:    $t \leftarrow t + 1$
17: **end while**
18: **return** $x^{bs}$
```

general framework of ACO metaheuristc for DOPs is described in Algorithm 11, where `DaemonActions` are the different pheromone modification strategies, described above, taken when a dynamic change occurs. The rest of the functions are the defined as in Algorithm 8.

6.5.2 Memory-based schemes

Guntsch and Middendorf [113] proposed an ACO algorithm with an associated memory to address dynamic environments, called the population-based ACO (P-ACO). Every iteration the best ants are stored in the memory which is used to update pheromone. The ants that enter the memory perform a positive update to their pheromone trails, whereas the ants that leave the memory perform a negative update to their pheromone trails. When a dynamic change occurs the information stored in the memory are repaired heuristically, and the pheromone trails are modified accordingly.

Algorithm 12 Population-based ACO

```
1: $t \leftarrow 0$
2: $P(0) \leftarrow$ InitializePopulation($\mu$)
3: InitializePheromoneTrails($\tau_0$)
4: $k_{long}(0) \leftarrow$ *empty solution*
5: $x^{bs} \leftarrow$ *empty*
6: **while** (termination condition *not* met) **do**
7:     $P(t) \leftarrow$ ConstructAntSolutions
8:     UpdatePopulation-List($k_{long}(t)$)
9:     DepositPheromoneTrails($k_{long}(t)$)
10:    $x^{ib} \leftarrow$ FindBest($P(t)$)
11:    **if** ($f(x^{ib}) < f(x^{bs})$) **then**
12:        $x^{bs} \leftarrow x^{ib}$
13:    **end if**
14:    **if** (environmental change is *detected*) **then**
15:        RepairHeuristically($k_{long}(t)$)
16:        GeneratePheromoneTrails($k_{long}(t)$)
17:    **end if**
18:    $t \leftarrow t + 1$
19:    $k_{long}(t) \leftarrow k_{long}(t-1)$
20: **end while**
21: **return** $x^{bs}$
```

Angus [3] proposed two variations of P-ACO, in which co-evolutionary techniques are applied to the memory. Fitness sharing P-ACO (FS-PACO) adjusts the fitness of the solutions stored in the memory according to their similarities, and the pheromone trails are adjusted accordingly. In simple crowding P-ACO (SC-PACO) the best ant replace the most similar ant in the memory. Both variations are able to maintain diversity because similar ants are eliminated. Furthermore, a memetic ACO (M-ACO) [180] was proposed, in which LS improvements are applied to the best ant before entering the memory.

Similarly with the approaches in the previous category, this category of approaches requires the detection of change. However, for this category there is a way to detect a change: if the solutions stored in the memory have a change in their fitness, it means that an environmental change has occurred [278]. The framework of P-ACO is described in Algorithm 12. Most of the functions in P-ACO are the same as the traditional ACO, described in Algorithm 11. However, additional functions have to be specified, in order to use the population-list denoted as $k_{long}(t)$, which are defined as follows:

- `UpdatePopulation-List`: updates $k_{long}(t)$ on iteration t using Equation 6.20 and 6.21.
- `RepairHeuristically`: repairs solutions stored in $k_{long}(t)$ in a way that causes the minimum cost to the resulted solution.
- `GeneratePheromoneTrails`: re-generates pheromone trails using the repaired solutions stored in $k_{long}(t)$.

6.5.3 Multi-colony algorithms

The aim of multi colony ACO algorithms is to have several ant colonies explore different areas in the search space and cooperate at some point by exchanging information or promote synchronization [208]. Parallel strategies have been first studied in [30, 242], where each colony uses its own pheromone matrix. Middendorf et. al [189] investigated several strategies in which different colonies exchange information.

There are two types of multi colony approaches, i.e., heterogeneous and homogeneous. Heterogeneous approaches have ant colonies working with different behaviours [100, 101], whereas homogeneous approaches have ant colonies working with the same behaviour [187, 188].

Although multi colony ACO performs better than single colony ACO in static environments [242], there are still no evidence regarding their performance in dynamic environments. However, this area is worth further investigation and can be considered as a future work.

6.6 Applications of ACO for Dynamic Combinatorial Optimization

Table 6.1 shows the important, or the most popular, ACO algorithms designed for different applications in dynamic environments. The applications are categorized into three main types: routing, network routing and scheduling. On the contrast to the wide range of ACO applications in static combinatorial optimization, in dynamic combinatorial optimization the applications are restricted to routing problems (including network routing). In this thesis, we consider the application of ACO for the TSP and VRP with dynamic environments.

6.6.1 Dynamic TSP (DTSP)

On the contrast to the static TSP described in Chapter 4, the dynamic TSP (DTSP) is more challenging, because there are a series of linked $\mathcal{NP}$-complete problem instances that need to be optimized over time. For the DTSP, we refer to two main variations that differ in the type of environmental changes. P-ACO [113], ACO with $Restart-strategy$ [111], $\tau-strategy$ [114] and $\eta-strategy$ [114] have been applied to the DTSP where the cities of the current problem instance are exchanged with cities from a spare pool. ACO-*shaking* [82] has been applied on the DTSP where the cost of cities' links increases/decreases to represent potential traffic. The ACO applications for the DTSP are described in detail below. For more details on how to generate DTSPs from static problem instances, see Chapter 7.

Structure. The structure for the DTSP with traffic factors is the same as the static TSP. For the DTSP with exchangeable cities, the problem instance is divided into the current pool and the spare pool. The current pool is where the algorithm optimizes, and the spare pool is used to exchange cities. In this way the size of the

Table 6.1: ACO applications for dynamic combinatorial problems categorized by their type and sorted chronologically.

Problem type	Problem name	Year	Main References
Routing	Travelling Salesperson	2001	[111, 114]
		2002	[12, 13, 82, 113]
		2005	[162]
		2006	[3]
		2011	[180, 181]
	Vehicle Routing	2003	[195]
		2005	[196]
		2012	[182]
Network Routing	Wired	1996	[228]
		1997	[71, 229]
		1998	[18, 72]
		2000	[75]
		2004	[35]
	Wireless	2000	[31]
		2001	[32]
		2002	[179, 234]
		2004	[73]
		2005	[74]
Scheduling	Job-shop	2007	[168]
		2010	[141]

problem instance remains the same.

Constraints. There are no additional constraints for the DTSPs.

Dynamics. The environmental changes of the DTSP with traffic factors occur by the increase of the cost of some links, and decreasing the previously traffic factors from the links. However, there is no evidence on how the magnitude of environmental changes is defined [82]. The environmental changes of the DTSP with exchangeable cities occur by exchanging the same number of cities from the current pool with those in the spare pool. The magnitude of change depends on the number of cities exchanged. The frequency of change for both DTSPs is defined by the algorithmic iterations.

Actions after change. In the P-ACO the solutions stored in $k_{long}(t)$ of limited size K_l are repaired using the *keep-elitist* strategy [113]. The offended cities are removed, and the cities from the spare pool are added to the positions that lead to the best fitness when they are re-evaluated. In the $Restart - strategy$, all the pheromone trails are re-initialized with an equal amount. In the $\tau - strategy$ and $\eta - strategy$ the pheromone trails of the offended cities are modified. The P-ACO and $Restart - strategy$ can be applied on both DTSPs, whereas $\tau - strategy$ and $\eta - strategy$ can be applied only to the DTSP with exchangeable cities. ACO-*shaking* smooths the pheromone trails using a logarithmic formula after pheromone evaporation, which is defined as follows:

$$\tau_{ij} \leftarrow \tau_0(1 + \log(\tau_{ij}/\tau_0)), \forall (i,j) \in E, \tag{6.18}$$

Solution construction. The solution construction for the DTSPs is the same as for static TSPs. All the algorithms, expect ACO-*shaking*, use the pseudorandom proportional rule in Equation 4.13.

Pheromone trails and heuristic information. The pheromone trails τ_{ij} and heuristic information η_{ij} are associated with the links of the problem instances as in the static TSP. However, when the links of the cities change due to traffic factor the heuristic information changes as well.

Pheromone update. ACO-*shaking* updates pheromone with the use of the elitist pheromone strategy of EAS defined in Equation 4.5. $Restart\!-\!strategy$, $\eta\!-\!strategy$ and $\tau - strategy$ uses another elitist strategy to update pheromone where one elite ant deposits pheromone along the best-so-far ant solution. For every city i a constant amount of pheromone is added to the link (i, j) when j is the successor of i in the best-so-far ant solution. The constant amount is defined as:

$$\tau_{ij} \leftarrow \tau_{ij} + \frac{1}{4}\rho. \tag{6.19}$$

P-ACO uses the population-list to update pheromone. Whenever ant k enters $k_{long}(t)$ a constant positive update is performed, defined as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}, \forall (i,j) \in T^k, \tag{6.20}$$

where $\Delta\tau_{ij} = (\tau_0 - \tau_{max})/K_l$ where K_l is the size of the population-list. When the population-list is full, an existing ant in $k_{long}(t)$, i.e., ant k', needs to be removed to make space for a new ant. Accordingly, a constant negative update is performed to ant k', defined as follows:

$$\tau_{ij} \leftarrow \tau_{ij} - \Delta\tau_{ij}, \forall (i,j) \in T^{k'}, \tag{6.21}$$

where $\Delta\tau_{ij}$ is defined as in Equation 6.20. There are many variations of the P-ACO algorithm that differ on the update policy of the population list:
1) The *Age*, *Quality*, *Prob* and *Age & Prob* update policies [113]. In the default Age strategy, the first ant that entered the population-list is replaced by the new ant which is described in Equations 6.20 and 6.21. In the Quality strategy, the new ant

replaced the worst ant in the population if it is better. In the Prob strategy, the new ant replaces an existing ant in the population-list, probabilistically, in which the ants with highest fitness have less chances to be replaced. The Age & Prob strategy, is a combination of the Age and Prob strategies described above, where Age is used for the removal of the ant from the population-list and Prob is used for the insertion of the new ant in the population-list. The difference of the hybrid Age & Prob strategy from the default Age strategy lies in the selection of the new ant that replaces the first ant that entered the population-list. In the former strategy the ant is selected probabilistically, whereas in the latter strategy the best ant is selected.

2) More update strategies for the population-list have been proposed in FS-PACO and SC-PACO, which are based on co-evolutionary techniques and the difference between ants [3], e.g., the similarity metric $S(p,q)$ defined in Equation 5.2. FS-PACO applies fitness sharing to the population-list, where each ant's solution value, say C^p for ant p, is de-rated as follows:

$$C^{p'} = \frac{C^p}{c_p}, \tag{6.22}$$

where $C^{p'}$ is the de-rated solution value and c_p is the niche count, defined as follows:

$$c_p = \sum_{q=1}^{K_l} sh\left(p,q\right), \tag{6.23}$$

where $sh(\cdot)$ is the sharing function defined as follows:

$$sh(p,q) = \begin{cases} 1 - \left(\frac{S(p,q)}{\sigma_{share}}\right)^a, \text{if } S(p,q) < \sigma_{share}, \\ 0, \qquad\qquad\qquad \text{otherwise}, \end{cases} \tag{6.24}$$

where σ_{share} is the niche radius and a is the parameter that determines the shape of the sharing function. The pheromone update is performed using the shared fitness, where all ants in the population-list are removed by the new ones. In SC-PACO simple crowding is applied, where ant p replaces the closest ant, e.g., q, in the population-list. In other words, the most similar ants are replaced using the $S(p, q)$ metric, if ant p is better than ant q. The pheromone update policy for SC-PACO and FS-PACO is the same as the default one defined in Equations 6.20 and 6.21, but with different $\Delta\tau_{ij} = \left(\frac{C^{bs}}{C^{p}}\right)^{\lambda}$, where C^{bs} is defined as in Equation 4.6, C^{p} is the solution quality of ant p, and λ is the parameter that determines the relative influence of pheromone quality. Note that all algorithms have pheromone evaporation as defined in Equation 4.2, except for the P-ACO, FS-PACO and SC-PACO algorithms, in which no evaporation is used.

3) Another variation of the P-ACO is the M-ACO [180] which follows the same framework, in addition with multiple LS improvements using blind inversions (BI) and guided inversions (GI). After constructing solutions, the best ant is selected to be improved by an LS operator before it enters the population-list. The LS operator is applied for several steps. In BI, the second object, which determines the size of the segment to be reversed, is selected randomly from the same individual of the first object. In GI, the second city is determined according to another individual randomly selected from the current population. The two inversion operators compete and cooperate in order to obtain the advantages of both of them during different periods when they are effective. Both BI and GI are selected probabilistically at every step of an LS operation on every iteration of the algorithm. Let p_{bi} and p_{gi} denote the probability of applying BI and GI to the individual selected for LS, respectively, where $p_{bi} + p_{gi} = 1$. Initially, the probabilities are both set to 0.5 in order to promote a fair competition between the two operators. The probabilities are adjusted according to the improvement each inversion operator has achieved on every LS step. The probability of the operator with the higher improvement is increased using a similar mechanism as introduced in [261]. Let ξ^{i} denote the degree

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

$$\xi^i = \frac{\left|C^{best'} - C^{best}\right|}{C^{best}}, \tag{6.25}$$

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

$$p_{bi}(t+1) = p_{bi}(t) + \xi^i_{bi}(t), \tag{6.26}$$

$$p_{gi}(t+1) = p_{gi}(t) + \xi^i_{gi}(t), \tag{6.27}$$

$$p_{bi}(t+1) = \frac{p_{bi}(t+1)}{p_{bi}(t+1) + p_{gi}(t+1)}, \tag{6.28}$$

$$p_{gi}(t+1) = 1 - p_{bi}(t+1), \tag{6.29}$$

where $\xi^i_{bi}(t)$ and $\xi^i_{gi}(t)$ are the total degree of improvement achieved by BI and GI operators at iteration t, respectively. Both p_{bi} and p_{gi} are set to their initial value, i.e., 0.5, when an environmental change occurs in order to re-start the cooperation and competition when a new environment arrives. Due to the high exploitation an LS provides, *triggered* random immigrants are added to the population-list whenever the population-list reaches a predefined threshold calculated by the diversity defined in Equation 6.9.

General comments. Recently, ACO has been integrated with immigrants schemes to address different DTSPs with traffic factors (we describe the proposed algorithms in great detail in Chapter 8).

6.6.2 Dynamic VRP (DVRP)

Similarly with the DTSP, the dynamic VRP (DVRP) is challenging because it requires continuous re-optimization of the changing optimum. ACO has been applied to the DVRP in which orders arrive incrementally. ACS-DVRP [195, 196], is the single colony version of MACS-VRP described in Chapter 4 and it is the only ACO algorithm applied to the DVRP with dynamic demands and to the DVRP in general. Recently, M-ACO [182], described in Section 6.6.1 for the DTSP, has been applied to solve the DVRP with traffic factors, with promising results. To the best of our knowledge, there is no other application of the ACO metaheuristic for the DVRP (a recent survey paper is available in [209]). The existing ACO applications for the DVRP are described in detail below. For more details on how to generate DVRPs from static problem instances, see Chapter 7.

Structure. The structure for the DVRP with dynamic demands is strongly related to the static VRP. The difference lies in that new orders (or customers) arrive when the working day has already started. The dynamic changes occur incrementally. More precisely, each static VRP will contain all the customers known at that time, but their demand is not satisfied yet. The structure of the DVRP with traffic factors is identical to the corresponding one of the DTSP, and will not be described again in this section.

Constraints. The constraints are the same as in the static VRP. An additional constraint is that customers become visible using time slice for the DVRP with dynamic demands.

Dynamics. The environmental changes of DVRP with dynamic demands occur when a new time slice will make available new customers. For example when a vehicle leaves the depot at time t_0 an initial route is generated to visit the currently known requests. While the vehicle executes its route, new requests appear at time t_1

and the initial route is adjusted to satisfy the demand requests of the new customers. Note that the dynamic changes appear in an incremental way. The environmental changes of the DVRP with traffic factors occur in the same way as in the DTSP with traffic factors.

Actions after change. The only action when a new time slice occurs is the *event manager*, which collects the new orders and keeps track of already visited customers.

Solution construction. The solution construction is the same as in MACS-VRP, which uses dummy depots to represent the number of vehicles. For M-ACO the solution construction is the same as in AS$_{rank}$-CVRP; see Section 4.6.2 for more details.

Pheromone trails and heuristic information. The pheromone trails τ_{ij} and the heuristic information η_{ij} are associated with the links of the problem instances as in the static VRP. The heuristic information of the algorithms is defined as the inverse of the distance or travel time between customers i and j (for more details see [196]).

Pheromone update. ACS-DVRP follows the same pheromone update policy with the MACS-VRP. An additional action is taken when a new time slice occurs, in which the pheromone trails generated from the old time slice are transferred to the new one. However, they are regulated as:

$$\tau_{ij} \leftarrow (1-\gamma)\tau_{ij} + \gamma\tau_0, \tag{6.30}$$

where γ is a parameter to smooth the intensity of the old pheromone trails. The new customers applied are initialized with τ_0. For M-ACO the same pheromone update policy as the corresponding algorithm in the DTSP is used. The only difference is in the LS operators used. In the DTSP, blind and guided inversions have been used, whereas in the DVRP blind and guided swaps have been used; see Algorithm 10 for more details. The diversity enhancement and the rule where multiple LS operators are applied remain unchanged.

General comments. Recently, ACO has been integrated with immigrants schemes to address a new variation of the DVRP with traffic factors (we describe the proposed algorithms in great detail in Chapter 8).

6.7 Theoretical Development

As discussed in Chapter 3, it is very challenging to analyze metaheuristics on SOPs and theoretical work is limited. Some main theoretical works done regarding (1+1) EA, an EA with population size 1, for static environments include first hitting time [124] and drifting analysis [123]. A recent survey regarding the impact of different components of EAs, including selection, mutation, crossover, parameter setting, and interactions among them can be found in [285]. In comparison with the theoretical work of metaheuristics for static optimization problems, the work for DOPs is even more limited.

In evidence that the research in dynamic optimization has been mainly made using EAs, all of the theoretical work developed for DOPs concern EAs. To the best of our knowledge, there is no any theoretical work for ACO for DOPs. In fact, the theoretical work for dynamic optimization extends the one made in EAs for static optimization. (1+1) EAs are used to a very simple problem, i.e., the dynamic bit-matching problem [25, 241]. Other EA theoretical developments for DOPs include the analysis of the frequency and magnitude of change in dynamic environments [225], the analysis of a benchmark generator for binary-encoded problems [250], the analysis of the dynamic subset sum problem [226], the analysis of the dynamic knapsack problem [26], and the analysis of the fitness landscape in dynamic environments [24, 134, 220, 221].

Due to the lack of theoretical work, algorithms in DOPs are analyzed empirically. Hence, benchmark generators for DOPs are essential tools for the performance evaluation of algorithms. A benchmark generator not only provides a standard method to compare algorithms, but also helps the development of new algorithms (more details in Chapter 7).

6.8 Summary

In this chapter we have presented the concept of dynamic optimization, which was mainly focused on EAs. Several strategies that enhance the performance of traditional EAs are defined. At the moment there is still no agreed measurement to evaluate the performance of EAs in dynamic optimization, and, thus, several performance measurements used are defined.

Similar to EAs, ACO suffers from the stagnation behaviour. Therefore, different strategies, mainly inspired from evolutionary computation, have been developed to delay convergence, increase diversity, and to transfer knowledge. The main applications and algorithmic contributions of ACO for DOPs are defined in this chapter. More details are given for the applications of DTSP and DVRP which are the problems used for the experiments in Chapter 8.

In general, most ACO applications assume static environment (see Table 4.1) and it can be observed that there are limited existing applications found under dynamic environments (see Table 6.1). However, it has been shown, from the limited applications available in DOPs, that ACO is a robust metaheuristic to accept the knowledge transferred from previous environments and adapt to the new one. Finally, the theoretical foundations of ACO in DOPs are even more weak, than the theoretical foundation of EAs in DOPs.

Chapter 7

Benchmark Generators for Dynamic Optimization Problems

7.1 Generating a Dynamic Environment

The field of dynamic optimization is related to the applications of nature-inspired algorithms [137]. The area is rapidly growing on strategies to enhance the performance of algorithms, but still there is limited theoretical work, as discussed in Section 6.7, due to the complexity of nature-inspired algorithms, e.g., EAs and ACO, and the difficulty to analyze them in the dynamic domain. Therefore, the development of benchmark problems to evaluate the algorithms empirically is appreciated by the evolutionary computation community. Such tools are not only useful to evaluate algorithms but also essential for the development of new algorithms.

A DOP can be otherwise defined as a series of several static instances. Hence, a straightforward method, but not efficient, to construct a dynamic test problem is to switch between different static instances that will cause an environmental change

[156]. The benchmark problems that generate dynamic environments following this methodology are specified for a single problem.

However, several general purpose dynamic benchmark generators have been proposed that re-shape the fitness landscape (a survey can be found in [49]). The most commonly used benchmark generators are: (1) Moving Peaks [20]; (2) DF1 [194]; and (3) exclusive-or (XOR) DOP [271]. The first two benchmark problems work for the continuous domain where they use functions with adjustable parameters to simulate shifting landscapes. The continuous space can be modelled as a "field of cones" [194], where each cone is adjusted individually to represent different dynamics. A similar approach is not feasible for the combinatorial space because the landscape is indistinct and cannot be defined without reference to the optimization algorithm, as described in Chapter 3.

7.2 Properties of Dynamic Benchmark Generators

In general, a benchmark can be defined as standard test problems designed for the development of new algorithms and the comparison with existing ones. A useful benchmark should have the following general properties [21, 193, 272, 293]:

- **Simplicity**: It should be simple and efficient in order to be easily adapted by other users.
- **Flexibility**: It should be flexible in order for the user to configure the different aspects of dynamism, i.e., frequency and magnitude of change, and periodicity as described in Chapter 6.

- **Generality**: It should be general enough to address a wide range of problems and be compatible with different types of metaheuristics.
- **Applicability**: It should model, to some extend, real-world situations.

Other characteristics or properties a benchmark generator for DOPs may have are classified as follows [200, 294]:

- **Known/Unknown Optimum**: Whether the environmental changes affect the optimum explicitly and cause it to change or the environmental changes shift the population to a different location in the fitness landscape but the optimum remains unchanged.
- **Predictable/Unpredictable Changes**: Whether the environment may change periodically by setting the frequency of change with a fixed number or the frequency of change may vary. In the former case the changes are predictable whereas in the latter case they are not.
- **Detectable/Undetectable Changes**: Whether the environmental changes can be detected, even if they are unpredictable, e.g., with detectors, or the environmental changes can not be detected because, apart from dynamic changes, noise is generated frequently.
- **Random/Cyclic Changes**: Whether the environmental changes occur in a random pattern or the environmental changes occur in a cyclic pattern. In the former case there is no guarantee that any previously generated environment will re-appear in the future, whereas in the latter case it is guaranteed.
- **Time-linkage/Non time-linkage**: Whether the dynamic changes in the future depend on the current or previous solutions found by the algorithm or not [19].

- **Dimensional/Non-Dimensional Changes**: Whether the environmental changes affect the problem size or not. Moreover, it should be clear which factors of the problem change, e.g., objective, constraints, domain variables, etc.

7.3 Benchmarks for Binary-Encoded Problems

The XOR DOP generator [271] is the only benchmark for the combinatorial space that constructs a dynamic environment from any static binary-encoded function $f(\vec{s})$, where $\vec{s} \in \{0,1\}^l$, by a bitwise XOR operator, where l is the size of vector $\vec{s}$, e.g., the encoding length of an individual. XOR DOP simply shifts the population of individuals into a different location in the fitness landscape. Hence, the global optimum remains known during the environmental changes if it is known for the base static function $f(\vec{s})$. It is the only benchmark generator widely accepted to compare algorithms in binary-encoded DOPs.

Therefore, every f iteration of an algorithm, a dynamic environment is generated as follows:

$$f(\vec{s}, t) = f(\vec{s} \oplus \vec{M}(T)), \tag{7.1}$$

where $\oplus$ is the exclusive-or operator, i.e., $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 0 = 0$, $T = \lceil t/f \rceil$ is the index of the period of change, t is the current count of iterations, and $\vec{M}(T) \in \{0,1\}^l$ is a binary mask of period T which is generated incrementally as follows:

$$\vec{M}(T) = \vec{M}(T-1) \oplus \vec{P}(T), \tag{7.2}$$

where $\vec{P}(T) \in \{0,1\}^l$ is a binary template randomly generated for period T that contains $m \times l$ ones, where m defines the magnitude of change. Initially, the mask $\vec{M}(0)$ contains only zeros that indicates no change in the environment. The constant parameters f and $m \in [0.0, 1.0]$ control the frequency and magnitude of change, respectively. XOR DOP is applied on each individual in the population. However, it is not feasible for permutation-encoded combinatorial problems, e.g. TSPs and VRPs.

By default, the environmental changes in XOR occur in a random pattern. In [276] XOR DOP has been extended to construct the environmental changes cyclically, and later on cyclically with additional noise [283]. In [278] the magnitude of change is not fixed, but it varies in every change.

7.4 Benchmarks for Permutation-Encoded Problems

Most research on dynamic optimization has been done with EAs on binary-encoded combinatorial problems. Recently, ACO algorithms have been found effective on permutation-encoded DOPs, e.g., the DTSP and its variants. However, in the case of permutation-encoded problems, in which the solution $f(x)$ is a set of numbers that represent a position in a sequence, researchers prefer their own benchmark problems to address different real-world applications.

Guntsch et al. [112] proposed a benchmark DTSP in which a number of cities are exchanged between the actual problem instance and a spare pool of cities. The same benchmark problem has been adapted in [180]. Eyckelhof and Snoek proposed the DTSP where the cost of the cities' arcs vary [82]. The same benchmark has been adapted in [181] to represent potential traffic. Younes et al. [293] introduced a benchmark DTSP with different modes, in which each mode introduces a different dynamic. Kilby et al. [147] proposed a benchmark for the DVRP where customer requests are revealed incrementally. The same benchmark has been adapted by Montemanni et al. [196].

On the other hand, Younes et al. [294] introduced a general benchmark framework that applies a mapping function on each permutation-encoded individual. The mapping function swaps the labels, i.e., the IDs, between two components and all the individuals in the population are treated on the same way. In this way, the individuals represent different solutions after a dynamic change but the fitness landscape of the problem instance does not change. However, this generator is restricted to the range of algorithms and problems that it is compatible with and it is limited to the accuracy regarding the magnitude of change.

Due to the high number of specialized benchmark generators for permutation-encoded problems, the development of a general one that converts the base of a static optimization problem to a dynamic, as XOR DOP for binary-encoded problems, is vital, since many of them are not available and they are difficult to be adapted. Moreover, it is impossible to know how close to the optimum the algorithm performs on each environmental change.

In the next sections, we describe two novel benchmark generators, where in the first one the optimum is known during the environmental changes, because it simply moves the population to a different location in the fitness landscape, and in the second one the optimum is unknown during the environmental changes, because it modifies the fitness landscape. More precisely, with the former benchmark generator it is possible to know how close to the optimum, if known, the algorithm performs at each environmental change, whereas with the latter benchmark generator it is impossible. The benchmark generators are used to generate DTSP and DVRP dynamic cases from any static problem instance. These benchmark generators are used later in Chapter 8 to evaluate the performance of our proposed algorithms.

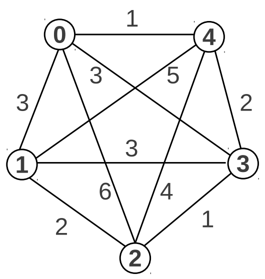
7.4.1 Known optimum

The proposed dynamic benchmark generator for permutation-encoded problems (DBGP) is designed in such a way to allow full control over the important aspects of dynamics and to convert the base of any benchmark static COP with known optimum to a dynamic one without causing the optimum to change. Such static instances can be obtained from the TSPLIB and VRPLIB, where most of the instances have been solved to optimality.

Some researchers want to observe "how close to the moving optimum a solution found by an algorithm is?". Probably it is the best way to evaluate the effectiveness of an algorithm for DOPs, in addition to the time needed to converge to that optimum. However, the global optimum is needed for every changing environment and this is very challenging due to the $\mathcal{NP}$-completeness of most COPs. Since a DOP can be considered as several static instances, a direct way is to solve each one to optimality, which may be non-trivial or even impossible, especially for large problem instances. It may be possible on small problem instances, but then it will reduce the usefulness of benchmarking. Hence, the need for a benchmark generator to address the challenges of comparison is increased, but it is even harder to develop a generator for DOPs with known optimum in COPs, without re-optimization.

The basic idea of the proposed DBGP is to modify the encoding of the problem instance, instead of the encoding of each individual, i.e., the distance matrix, without affecting its fitness landscape. To illustrate such a dynamic change, let $G = (V, E)$ be a weighted graph where V is a set of n nodes, each of which has a location defined by (x, y), and E is a set of arcs. Each arc (i, j) is associated with a non-negative weight d_{ij}. Usually, the distance matrix of a problem instance is defined as $\vec{D} = (d_{ij})_{n \times n}$, where n is the size of the problem instance. Then, an environmental change may occur at any time by swapping the location of some node i with the location of

Optimum -> (0,4,3,2,1,0) = **9**

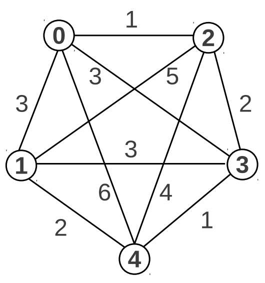


Distance matrix before change

	0	1	2	3	4
0	0	3	6	5	1
1	3	0	2	3	3
2	6	2	0	1	4
3	5	3	1	0	2
4	1	3	4	2	0

Swap City Location (4,2)

Optimum -> (0,2,3,4,1,0) = **9**



Distance matrix after change

	0	1	2	3	4
0	0	3	1	5	6
1	3	0	3	3	2
2	1	3	0	2	4
3	5	3	2	0	1
4	6	2	4	1	0

FIGURE 7.1: Illustration of the distance matrix with the optimum solution of the problem instance before and after a dynamic change.

some node j. In this way, the values in the distance matrix are reallocated but the optimum remains the same; see Figure 7.1.

The dynamic environments constructed by DBGP may not reflect a real-life situation but achieve the main goal of a benchmark in which the optimum is known during all the environmental changes. In other words, DBGP sacrifices the realistic modelling of application problems for the sake of benchmarking. Moreover, it is simple and can be adapted to any TSP and its variants to compare algorithms in dynamic environments.

7.4.1.1 Frequency and Magnitude of Change

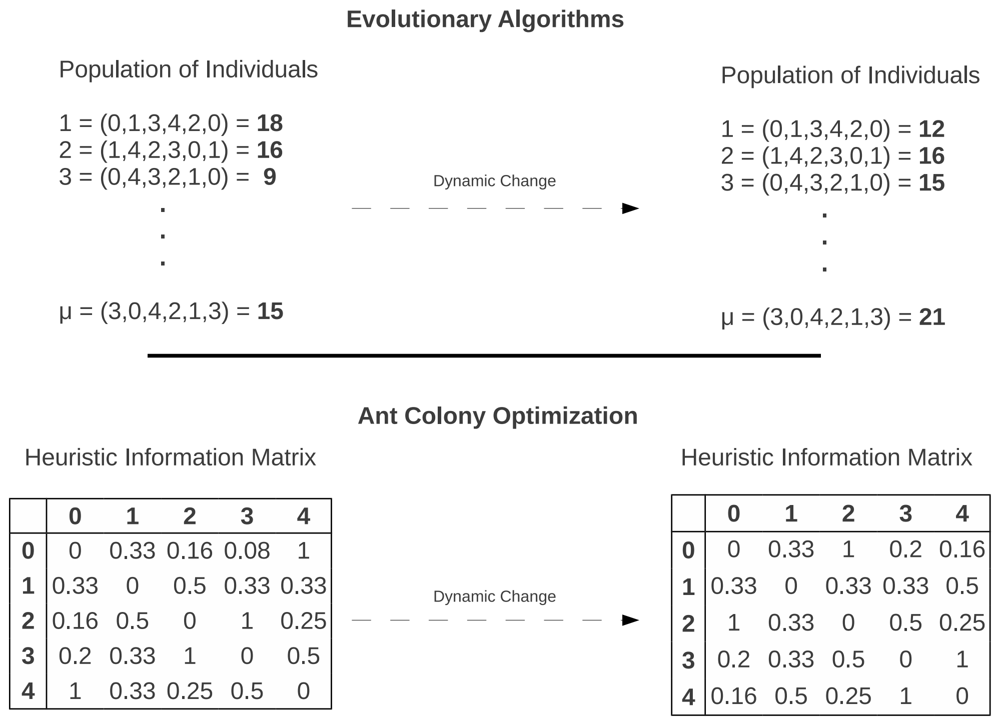
Every f iteration a random vector $\vec{r}_V(T) = (1, \ldots, n)$ is generated that contains all the components of a problem instance of size n, where $T = \lceil t/f \rceil$ is the index of the period of change, and t is the iteration count of the algorithm. For example, for the TSP the components are the cities and for the VRP they are the customers, where both types of components have a location that is defined by (x, y). The magnitude m of change depends on the number of swapped locations of components.

More precisely, $m \in [0.0, 1.0]$ defines the degree of change, in which only the first $m \times n$ of $\vec{r}_V(T)$ object locations are swapped. In order to restrict the swaps to the first components, a randomly reordered vector $\vec{r}_{V'}(T)$ is generated that contains the first $m \times n$ components of $\vec{r}_V(T)$. Therefore, exactly $m \times n$ pairwise swaps are performed using the two random vectors starting from the first pair. In Younes' generator, the magnitude of change is expressed as the number of swaps imposed on the mapping function. In this way, the components affected from the dynamic change may not correspond to the predefined magnitude parameter. For example, if $m = 0.5$, half of the components may be swapped with the remaining half of the components of the optimization problem. Hence, the change affects all the components and may be considered as $m = 1.0$.

Moreover, the frequency of change is defined by the constant parameter f which is usually defined by the algorithmic iterations. Hence, for every f iteration $m \times n$ pairwise swaps are performed with probability. However, before each environmental change, the previous pairwise swaps are reversed, starting from the end of $\vec{r}_V(T-1)$ and $\vec{r}_{V'}(T-1)$. In this way, the environmental changes are always applied to the encoding of the initial stationary problem instance.

7.4.1.2 Effect on the Algorithms

DBPG can be applied to algorithms that either maintain an actual population or not, because the dynamic changes occur to the encoding of the actual problem instance. In this way, the solutions of EAs have the same encoding as before a dynamic



	0	**1**	**2**	**3**	**4**
0	0	0.33	0.16	0.08	1
1	0.33	0	0.5	0.33	0.33
2	0.16	0.5	0	1	0.25
3	0.2	0.33	1	0	0.5
4	1	0.33	0.25	0.5	0

	0	**1**	**2**	**3**	**4**
0	0	0.33	1	0.2	0.16
1	0.33	0	0.33	0.33	0.5
2	1	0.33	0	0.5	0.25
3	0.2	0.33	0.5	0	1
4	0.16	0.5	0.25	1	0

FIGURE 7.2: Illustration of the effect on the population of EAs and the heuristic information used from ACO before and after a dynamic change.

change, but have a different cost after a dynamic change. On the other hand, the constructive procedure of ACO is affected since different heuristic information is generated whereas the pheromone matrix remains unchanged. See Figure 7.2 for more details.

In general, DBGP shifts the population of EAs and biases the population of ACO algorithms to a new location in the fitness landscape of the search space, respectively. Younes' generator assumes that the solver has a population of solutions since the mapping function is applied on the encoding of each individual, e.g., EAs. Hence, it cannot be applied to algorithms that do not maintain an actual population, e.g., ACO.

Another advantage of the DBGP against Younes' generator [294] is that in the VRP the solutions after a dynamic change may represent an infeasible solution when dealing with EAs. This is because when the label of the customer changes then its demand changes, and the capacity constraint is possible to be violated. Hence, a repair operator or a penalty function has to be applied. The proposed DBGP overcomes this problem since only the location of customers changes whereas the label and the demand remain unchanged.

7.4.1.3 Cyclic Dynamic Environments

The default dynamic environments generated by DBGP do not guarantee that any of the previously generated environment will re-appear. Such environments are called *random dynamic environments*; see Figure 7.3(a). In fact, some algorithms that are enhanced with memory are expected to work better on dynamic environments that re-appear in the future [275]. Such environments are called *cyclic dynamic environments*; see Figure 7.3(b), and they can be generated as follows. First, we generate K random vectors $(\vec{r}_V(0), \ldots, \vec{r}_V(K-1))$ with their corresponding reordered vectors as the base states in the search space. Initially, the first base state is applied. Then, every f iterations the previous dynamic changes are reversed, starting from the end of random vector, and then the new ones are applied from the next base state. In this way, it is guaranteed that the environments generated from the base states will re-appear.

DBGP has two options for cyclic dynamic environments regarding the way the base states are selected: (1) cyclic, where the base states are selected as in a fixed logical ring; and (2) randomly, where the base states are selected randomly.

From the above cyclic environment generator, we can further construct cyclic dynamic environments with noise as follows. Each time a new base state is to be selected, swaps are performed from the components that are not in $\vec{r}_V(T)$ with a

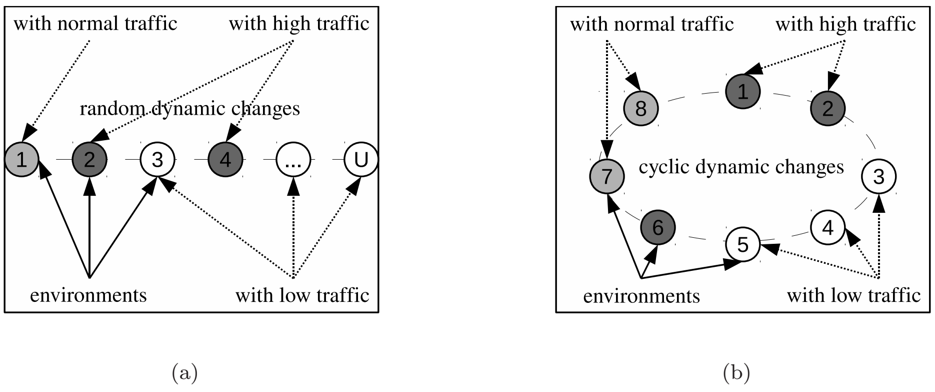


FIGURE 7.3: Illustration of a random dynamic environment with unlimited states and a cyclic dynamic environment with 8 states. Each node represents a different environment where white, light grey, and dark grey, represent low, medium, and high traffic jams, respectively.

small probability, i.e., p_{noise}. Note that the swaps due to the noise are reversed in the same way as with the dynamic changes above.

7.4.1.4 Varying f and m Parameters

In the random and cyclic environments described above the f and m parameters remain fixed during the execution of the algorithm. An additional feature of DBGP is to vary the values of f and m with a randomly generated number with a uniform distribution in $[1, 100]$ and $[0.0, 1.0]$, respectively, for each environmental change. Note that in this type of environment the changes are not reversed as in the random and cyclic environments.

7.4.2 Unknown optimum

The basic idea of this benchmark generator is to increase or decrease the cost of the arcs in E regularly in time, in order to represent potential traffic in roads; see Figure 7.3. Similar to DBGP, this benchmark generator can convert the base of any benchmark static COP to a dynamic one and allows full control over the important aspects of dynamics. However, it causes the optimum to actually change since the values of the distance matrix $\vec{D} = (d_{ij})_{n \times n}$ are modified with traffic factors.

7.4.2.1 Frequency and Magnitude of Change

A dynamic environment with traffic factors is generated as follows. We assume that the cost of arc (i, j) is $d'_{ij} = d_{ij} \times c_{ij}$, where d_{ij} is the normal travelled distance and c_{ij} is the traffic factor between components i and j. Every f iterations of algorithmic iterations, a random number R in $[F_L, F_U]$ is generated probabilistically to represent the traffic factor between components, where F_L and F_U are the lower and upper bounds of the traffic factor, respectively. Each arc has a probability m to add traffic by generating a different R value every f iteration, where the traffic factor c_{ij} of the remaining arcs is set to 1, which indicates no traffic.

For example, a dynamic case with high traffic is constructed by setting traffic factor values closer to F_U with a higher probability to be generated, while for a dynamic case with low traffic, a higher probability is given to traffic factor values closer to F_L.

7.4.2.2 Effect on the Algorithms

Similarly with DBGP, it can be applied to algorithms that either maintain an actual population or not, because the dynamic changes occur to the encoding of the actual problem instance.

The difference in this benchmark generator is that it changes the fitness landscape because the traffic factors increase the values in the distance matrix. In DBGP, the values in the distance matrix are swapped but they are not modified.

7.4.2.3 Cyclic Dynamic Environments

Another variation of a dynamic environment is where changes occur with a cyclic pattern. In other words, previous environments will appear again in the future. Such environments are more realistic since they represent a 24-hour traffic jam situation in a day.

A cyclic environment can be constructed by generating different dynamic cases with traffic factors as the base states, representing environments where each arc has a probability m to add low, normal, or high traffic as in random dynamic environments. Then, the environment cycles among these base states, every f iterations, in a fixed logical ring as represented in Figure 7.3(b). Depending on the period of the day, dynamic cases with different traffic factors can be generated. For example, during the rush hour periods, a higher probability is given to the traffic factors closer to F_U, whereas during evening hour periods, a lower probability is given to F_U and a higher probability to F_L.

7.4.2.4 Varying f and m Parameters

Dynamic environments with varying m and f are generated in the same way as in DBGP.

7.5 Summary

In this chapter we have presented two benchmark generators for dynamic permutation problems (TSP and VRP). In general, both benchmark generators can be applied to a wide range of permutation-encoded problems and can be used with different types of algorithms. Moreover, they both satisfy some of the general properties of a good benchmark generator: simplicity because they are simple methods and can be adopted easily; flexibility because they allow full control of the frequency and magnitude of change parameters, and the periodicity of the changes; and generality because they can address a wide range of algorithms and permutation-problems. For the case of applicability, DBGP does not satisfy this property whereas the other benchmark generator does, since it models potential traffic on the roads.

However, DBGP satisfies another characteristic where the optimum is known on each dynamic change, because it does not modify the fitness landscape of the problem, but it shifts the algorithm's population to a new location, whereas the other benchmark generator modifies the fitness landscape. On the other hand, both generators can generate predictable and unpredictable changes, since the frequency of change can be fixed or varied. Moreover, both can generate random and cyclic environments. Regarding the detection of changes, both benchmarks have detectable and undetectable changes. The type of changes applied can be considered as non-dimensional because the size of the problem remains the same.

Finally, none of the benchmark generators satisfies the time-linkage property. It is a good direction of future work and it deserves further development (for more information see Chapter 9).

Chapter 8

ACO with Immigrants Schemes

8.1 Motivation

Immigrants schemes have been successfully applied to GAs to solve different binary-encoded DOPs as described in Chapter 6. Immigrants schemes are simple to implement and integrate with a metaheuristic. Considering the results with conventional GAs, immigrants schemes are effective because they enhance GA's performance significantly [278]. In contrast, ACO has been mainly applied to permutation-encoded DOPs. An interesting approach is the P-ACO (see Algorithm 12), which has similarities with a GA because it maintains an actual population of solutions, i.e., $k_{long}(t)$. In conventional ACO algorithms the information of the population is stored in the pheromone trails.

Considering the good performance of immigrants schemes with GAs, and the similarities of P-ACO with GAs, then a direct approach to enhancing the performance of ACO algorithms in permutation-encoded DOPs is to integrate immigrants schemes with them. In fact, other techniques adapted from GAs, such as fitness sharing and simple crowding, have been integrated with the P-ACO in dynamic and multi-objective environments with promising results [3].

The immigrants schemes can maintain the diversity through the execution of ACO algorithm and hence avoid stagnation behaviour. Conventional ACO algorithms have pheromone evaporation that eliminate pheromone trails from solutions that are not useful, and may help them to adapt to the new environment. However, the adaptation may depend on the problem instance and the degree of change. On the other hand, P-ACO does not have pheromone evaporation, since it eliminates old solutions directly when they are removed from the population-list.

8.2 Description of Immigrants Schemes

8.2.1 The Framework

As mentioned before, the proposed framework of ACO algorithms with immigrants schemes is inspired from the GA characteristics of the P-ACO framework and the good performance of immigrants schemes in GAs for DOPs. Considering that P-ACO maintains a population of solutions, immigrant ants can be generated and replace ants in the current population. The proposed framework is able to maintain the diversity within the population and transfer knowledge from previous environments to the pheromone trails of the new environment. The main idea is to generate the pheromone information for every iteration of running the algorithm, considering information from the previous environment and extra information from the immigrants schemes.

Therefore, instead of using a long-term memory as in P-ACO, a short-term memory, denoted as $k_{short}(t)$, is used, where all ants stored from iteration $t-1$ are replaced by

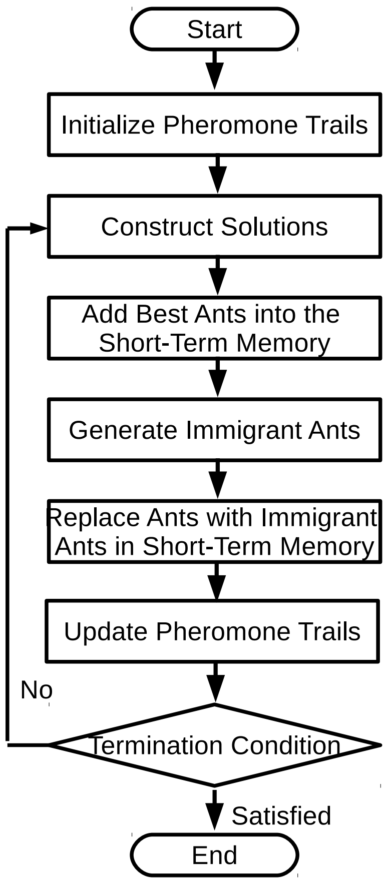

Start
Initialize Pheromone Trails
Construct Solutions
Add Best Ants into the Short-Term Memory
Generate Immigrant Ants
Replace Ants with Immigrant Ants in Short-Term Memory
Update Pheromone Trails
No
Termination Condition
Satisfied
End

FIGURE 8.1: Flowchart of the general framework of ACO algorithms with immigrants schemes.

the K_s best ants of the current iteration t, where K_s is the size of $k_{short}(t)$. Moreover, a number of immigrant ants are generated and replace the worst ants in $k_{short}(t)$. A flowchart of the framework is presented in Figure 8.1

The benefits of using $k_{short}(t)$ are closely related to the survival of ants in a dynamic environment, where no ant can survive in more than one iteration. For example, in iteration t, if ants are stored from iteration $t-2$ and an environmental change occurred in iteration $t-1$, then the solutions may not be feasible for the current environment in iteration t, and hence need to be repaired as in the P-ACO. Usually, a repair procedure is computationally expensive, and requires prior knowledge of the problem. Furthermore, this action can be taken only if the environmental changes can be detected, which sometimes is not applicable in real-world applications. For example, the traditional $\mathcal{MM}$AS algorithm with a re-initialization of pheromone trails when a dynamic change occurs may not be a sufficient choice on DOPs where the frequency of change is not available beforehand or the dynamic environment is noisy. Therefore, the pheromone evaporation is responsible to help the population of ants adapt in changing environments, as discussed previously.

8.2.2 Construct Solutions

The random proportional rule described in Equation 4.1 is used to construct solutions where, initially, all ants are randomly placed on objects, i.e., cities for the TSP or the depot for the VRP, and each ant represents a partial solution. A complete solution is constructed by adding to the partial solution objects which are selected probabilistically by each ant, considering pheromone trails and heuristic information.

Typically, the heuristic information η_{ij} is defined as $\eta_{ij} = 1/d_{ij}$, where d_{ij} is the distance between two objects i and j. For dynamic environments generated from the benchmark generator described in Chapter 7, the heuristic information includes the traffic factor added to the links of the objects such that, $\eta_{ij} = \frac{1}{(d_{ij} \times c_{ij})}$, where c_{ij} is the traffic factor.

8.2.3 Pheromone Update

The pheromone update differs from both traditional ACO and P-ACO. Every iteration t, the pheromone trails are associated with $k_{short}(t)$ and any change to the solutions in $k_{short}(t+1)$ reflects the pheromone trails. For example, when the worst ants are replaced by immigrant ants, the pheromone trails of each worst ant are removed, as follows:

$$\tau_{ij} \leftarrow \tau_{ij} - \Delta\tau_{ij}^{k}, \forall\ (i,j) \in T^{k}, \tag{8.1}$$

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

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{k}, \forall\ (i,j) \in T^{k}, \tag{8.2}$$

where $\Delta\tau_{ij}^{k}$ and T^k are as defined in Equation 8.1. This mechanism keeps the pheromone trails between a certain value τ_{min}, which is equal to τ_0, and a τ_{max} value, which can be calculated by $\tau_0 + \sum_{k=1}^{K_s} \Delta\tau_{ij}^{k}$. We have seen previously the importance to keep the pheromone trails to a certain level in the best performing variations of the AS algorithm, i.e., $\mathcal{MM}AS$ and ACS [66, 244].

The same pheromone policy is used when the ants from $k_{short}(t)$ are replaced from the new best K_s ants in iteration $t+1$ (see Equations 8.1 and 8.2).

8.2.4 Increase and Maintain Diversity

8.2.4.1 Random Immigrants ACO (RIACO)

Random immigrants have been found to perform well with GAs for different DOPs since they maintain a certain level of diversity during the execution [107, 274]. The principle is to introduce new randomly generated individuals and replace a small portion in the current population. RIACO follows the framework described above where $k_{short}(t)$ is used instead of $k_{long}(t)$. All the ants of the current iteration replace the old ones, instead of only replacing the oldest one as in P-ACO. Therefore, when ants are removed, a negative update is made to their pheromone trails as in

Algorithm 13 RIACO and EIACO

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

Equation 8.1, and when new ants are added, a positive update is made to their pheromone trails as in Equation 8.2.

However, before the pheromone trails are updated, a set S_{ri} of $r \times K_s$ immigrants are randomly generated to replace other ants in $k_{short}(t)$, where r is called the replacement rate. The pseudo-code of RIACO is presented in Algorithm 13. More precisely, to define RIACO the following functions have to be specified:

- `InitializePopulation`: generates the initial population with μ ants.

- `InitializePheromoneTrails`: initialize all pheromone trails with τ_0.
- `ConstructAntSolutions`: all ants construct feasible solutions.
- `AddBestAnts`: adds the K_s best ants from $P(t)$ to $k_{short}(t)$.
- `UpdatePheromone`: updates pheromone trails using Equations 8.1 and 8.2.
- `GenerateRandomImmigrants`: generates $r \times K_s$ random immigrants.
- `ReplaceAntsWithImmigrants`: replaces the worst ants in $k_{short}(t)$ with the generated immigrant ants.
- `FindBest`: returns the best ant in the current population for generation t, denoted as x^{ib}.

A random immigrant represents a feasible solution of the problem which is constructed randomly. For the DTSP, a random immigrant is generated by adding an unvisited city until all cities are used in order to represent one feasible TSP solution, in which at the end the first city is also added, which represents the returning of the salesperson to its home city. For the DVRP, a random immigrant ant is generated as follows. First, the depot is added as the starting point; then, an unvisited customer is randomly selected as the next point. This process is repeated until the first segment (starting from the most recent visit to the depot) of customers do not violate the capacity constraint. When the capacity constraint is violated the depot is added and another segment of customers starts. When all customers are visited the solution will represent one feasible VRP solution.

It is claimed that "the continuous adaptation of algorithms makes sense only when the environmental changes of a problem are small to medium" [20]. This is due to the fact that a new environment has a high chance to be similar with the old one. After a change occurs, transferring knowledge from the old environment to the pheromone trails may move the ants into promising areas in the new environment. Considering this argument, RIACO is expected to perform well in fast and significantly changing environments, since knowledge is not transferred by immigrants and the diversity is generated randomly.

8.2.4.2 Elitsm-Based Immigrants ACO (EIACO)

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

In order to generate guided diversity, EIACO is proposed to address DOPs by transferring knowledge from the previous environment. For each iteration t, within EIACO, the elite ant from the previous environment, i.e., the best ant from $k_{short}(t-1)$, is used as the base to generate a set S_{ei} of $r \times K_s$ elitism-based immigrants. For the DTSP, an elitism-based immigrant is generated by applying the inver-over operator [116] on the best ant, where two cities are selected and the segment between them is reversed. Similarly, for the DVRP, the inver-over operator is applied after the depot components of the best ant are removed. When the inversion operator finishes, the depot components are added so that the capacity constraint is satisfied in order to represent one feasible VRP solution which is similar with the best ant obtained from $k_{short}(t-1)$.

The pseudo-code of EIACO is also shown in Algorithm 13. All the functions that define EIACO are the same with RIACO except the `GenerateGuidedImmigrants`,

Algorithm 14 GenerateGuidedImmigrant(x^{elite})

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

which is presented in Algorithm 14. More precisely, to define the generation of a guided immigrants the following functions have to be specified:

- `SelectRandomCity`: selects a random component from the elite ant, denoted as x_r.
- `SelectNewRandomCity`: selects a second random component (different from the previous one) from the elite ant, denoted as x'_r, with probability 0.02.
- `SelectRandomAnt`: selects a random ant from the current population, denoted as x^{ran}.
- `NextCity`: assigns to x'_r the next component to the component x_r in the randomly selected ant.
- `Inversion`: inverses the components from the next component of component x_r to component x'_r in the elite ant, denoted as $x^{elite'}$.

This operator performs several inversions in which the size of the segment of each inversion is adapted by other ants (higher probability), and the number of inversions are terminated when the next or previous component of component x_r for $x^{elite'}$ is x'_r.

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

8.2.4.3 Memory-Based Immigrants ACO (MIACO)

MIACO is a generalized version of EIACO since not only the best ant from the previous environment is considered, but the best ant among several environments is considered as the base to generate immigrants in MIACO. The only difference between MIACO and EIACO lies in that MIACO uses both $k_{short}(t)$ and $k_{long}(t)$, where the first type of memory is updated and used as in RIACO and EIACO. The second type of memory is initialized with random ants (or memory points) and updated by replacing any of the randomly initialized ants, if they still exist in the memory, with the best-so-far ant; otherwise, the closest ant in the memory is replaced with the best-so-far ant if the best-so-far ant is better. Note that the update strategy of $k_{long}(t)$ in MIACO is different from that in P-ACO regarding which ant to replace, since in MIACO the most similar memory updating strategy is used [20], whereas in the default P-ACO, the new ant replaces the oldest one.

Algorithm 15 MIACO

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

In MIACO, a metric to define how close ant p is to ant q is used. For the case of DTSP it is defined as in Equation 5.2 and for the DVRP as in Equation 5.3. Apart from which ant is replaced in $k_{long}(t)$, the update strategy of MIACO is different from the one used in P-ACO with respect to when an ant is replaced. In P-ACO, the update occurs every iteration, whereas in MIACO the update occurs whenever a dynamic change occurs in order to store useful solutions from different environments. Therefore, for each iteration t within MIACO, the ants in $k_{long}(t)$ are re-evaluated

Algorithm 16 UpdateMemory($k_{long}(t)$)

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

in order to be valid with a new environment and to detect an environmental change. The ants in $k_{long}(t)$ are used as detectors; see Section 6.3 for more details. Then, the best ant from $k_{long}(t)$ is selected and used as the base to generate a set S_{mi} of $r \times K_s$ memory-based immigrants using Algorithm 14. Note that the memory-based immigrants for both the DTSP and DVRP are generated in the same way as the elitism-based immigrants, but differ on the ant used as the base to generate immigrants. For the former method the best ant from $k_{long}(t)$ is used, whereas for the latter method the best from the $k_{short}(t-1)$ is used.

The pseudo-code of MIACO is presented in Algorithm 15. All the functions that define MIACO are the same with RIACO and EIACO, but it contains an additional function that uses $k_{long}(t)$, called `UpdateMemory`, which is described in Algorithm 16. More precisely, the functions used in the memory's update procedure are defined as follows:

- `FindBest`: returns the best ant in the current population for generation t, denoted as x^{best}.

- `ReplaceRandomWithBest`: replaces a random ant, generated in the initialization phase and still exists, in $k_{long}(t)$ with the best ant previously selected, i.e., x^{best}.

- `FindClosest`: returns the most similar of ant of x^{best} in k_{long} using Equation 5.2, denoted as x^{cm}.

As discussed in Section 6.3, this mechanism is not suitable for dynamic environments with noise. Therefore, the update does not depend only on the detection of dynamic changes since in some real-world applications it is not easy or impossible to detect changes. For example, in DTSPs, noise may be added in every iteration of the algorithm apart from the traffic factor, which may also indicate environmental changes using the detection mechanism with the detectors from k_{long}. As a result, the algorithm will not be able to distinguish whether the change of the fitness in a solution is because of noise or an environmental change. The detection mechanism will not work properly because it will detect changes in every iteration due to the noise. Therefore, instead of updating $k_{long}(t)$ only in a fixed time interval, e.g., every f iterations, which is dependent on the dynamic changes, $k_{long}(t)$ is also updated in a dynamic pattern as presented in Algorithms 15 and 16. More precisely, on every update of $k_{long}(t)$, a random number $R \in [5, 10]$ is generated, which indicates the next update time. For example, if the memory is updated at iteration t, the next update will occur at iteration $t_M = t + R$ [283] (if no change is detected before iteration t_M).

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

MIACO may be beneficial on the same environmental cases with EIACO since it is a generalized version of EIACO. However, it may be also advantageous in cases where the previous environments will re-appear in the future, e.g., cyclic dynamic environments.

8.3 Experiments for the DTSP and DVRP

8.3.1 Experimental Setup

For the experiments in the DTSP and DVRP we have used the benchmark generator with unknown optimum described in Chapter 7, which models a real-world scenario, i.e., traffic on road networks. Therefore, from some (three) stationary benchmark problem instances with different sizes, several dynamic environments with different properties are generated. In all the following experiments each algorithm performs 1000 iterations for 30 independent runs on the same environmental changes, and the offline performance defined in Equation 6.3 is used. For the statistical results, Wilcoxon rank-sum test at a 0.05 level of significance is used.

In order to evaluate the adaptation and searching capabilities of each algorithm the frequency of change was set to $f = 5$, $f = 50$ and $f = 100$, indicating fast to slowly changing environments, respectively, and the magnitude of change was set to $m = 0.1$, $m = 0.25$, $m = 0.5$ and $m = 0.75$, indicating slightly, to medium, to significantly changing environments, respectively. As a result, for each of the three problem instances, 12 dynamic environments were generated, i.e., 3 f values $\times$ 4 m

values. The basic results for random and cyclic DTSPs are presented in Sections 8.3.2.5 and 8.3.2.6, respectively, and for random and cyclic DVRPs in Sections 8.3.3.3 and 8.3.3.4, respectively.

8.3.2 Experimental Results for the DTSP

8.3.2.1 Parameters Settings

The parameter settings of the algorithms differ from the ones used in the static TSP. Appendix C presents all the parameters used to the DTSP. Most of them have been inspired from the literature [3, 111, 113, 114] and [69, p. 71], whereas the important algorithmic parameters have been optimized via experiments as presented in the next subsections.

8.3.2.2 Effect of the Evaporation Rate ρ

For the results shown in Table 8.1, one of the best performing traditional ACO algorithm, i.e., $\mathcal{MM}$AS, is applied to slowly changing environments with different magnitudes, and the evaporation rate is optimized (RIACO, EIACO and MIACO do not have pheromone evaporation). In the static TSP, the recommended evaporation rate for $\mathcal{MM}$AS is $\rho = 0.02$ [69, p. 71], whereas in the DTSP it has the worst results.

It can be observed that as the evaporation rate increases up to 0.6 the solution quality improves, whereas when it is higher than 0.6 the solution quality is usually degraded or it has similar performance. As we discussed in Chapters 1 and 6, pheromone evaporation is an important mechanism in ACO algorithms to adapt in dynamic changes. This is because it eliminates pheromone trails generated from the

Table 8.1: Offline performance of $\mathcal{MM}$AS with different evaporation rates for the DTSP. Bold values indicate the best results.

$f = 100, m \Rightarrow$	0.1	0.25	0.5	0.75
Alg. & Inst.		kroA100		
$\mathcal{MM}$AS($\rho = 0.02$)	24591.19	27082.79	34861.35	47782.55
$\mathcal{MM}$AS($\rho = 0.2$)	23575.57	25858.26	33162.43	45547.84
$\mathcal{MM}$AS($\rho = 0.4$)	23367.60	25675.32	**32804.83**	45052.21
$\mathcal{MM}$AS($\rho = 0.6$)	23331.39	**25541.11**	32864.74	44994.17
$\mathcal{MM}$AS($\rho = 0.8$)	**23320.99**	25595.99	32845.60	**44986.17**
Alg. & Inst.		kroA150		
$\mathcal{MM}$AS($\rho = 0.02$)	31840.23	35648.34	43194.10	59544.27
$\mathcal{MM}$AS($\rho = 0.2$)	30259.85	34139.32	41192.83	57139.14
$\mathcal{MM}$AS($\rho = 0.4$)	30031.67	33854.66	**40809.19**	**56544.39**
$\mathcal{MM}$AS($\rho = 0.6$)	29938.17	33788.31	40832.21	56597.26
$\mathcal{MM}$AS($\rho = 0.8$)	**29913.39**	**33734.48**	40979.44	56882.78
Alg. & Inst.		kroA200		
$\mathcal{MM}$AS($\rho = 0.02$)	36037.17	39797.21	48778.72	69179.75
$\mathcal{MM}$AS($\rho = 0.2$)	34335.07	38100.82	46662.05	66448.51
$\mathcal{MM}$AS($\rho = 0.4$)	33920.44	37801.86	**46333.89**	**65742.27**
$\mathcal{MM}$AS($\rho = 0.6$)	**33741.35**	**37775.56**	46412.48	65878.12
$\mathcal{MM}$AS($\rho = 0.8$)	33741.89	37897.85	46712.78	66222.85

previous environment that are useless in the new environment, and may bias ants towards non-promising areas in the search space.

On the other hand, an extremely high evaporation rate decreases the performance because it may eliminate useful pheromone trails from the previous environment fast, and the knowledge transferred from the pheromone trails is destroyed quickly.

8.3.2.3 Effect of Short-Term Memory Size K_s

For the results shown in Table 8.2 and Figure 8.2, the proposed framework (without immigrant ants) is applied to slowly changing environments with different magnitudes of change, and the size of k_{short} memory is optimized. Differently from $\mathcal{MM}$AS, in our proposed framework there is no pheromone evaporation. It is a similar framework with the P-ACO described in Algorithm 12 regarding the pheromone update policy, but differs in the way memory is maintained. P-ACO uses a long-term memory because ants from more than one iteration are stored in the memory, whereas in our framework only ants from the previous iteration are stored.

When $K_s = 1$, the algorithm is actually a P-ACO with $K_l = 1$, which can be seen as an ACO algorithm with a global restart, since the pheromone trails are re-initialized in every iteration. It can be observed that when $K_s = 6$ the performance of the algorithm is improved, probably because the pheromone trails get more information from k_{short}. When an extreme value is given to K_s, which is closer to the actual population size, then the algorithm has no selection of the best ants. This is due to the fact that all the ants in the framework deposit a constant amount of pheromone and, thus, no effort is given to the best ants.

$\mathcal{MM}$AS and the proposed framework with $r = 0.0$ have been applied on the same dynamic environments. Therefore, when we compare the best values obtained from $\mathcal{MM}$AS and the proposed framework, there is an interesting observation; see Tables 8.1 and 8.2. On kroA100 with $m = 0.1$ and $m = 0.25$, the performance values of $\mathcal{MM}$AS($\rho = 0.6$) are 23331.39 and 25541.11, respectively, whereas the corresponding values of the proposed framework with $K_s = 6$ are 23414.59 and 25676.11, respectively, which means that the former one performs better than the latter one for both dynamic test cases.

Table 8.2: Offline performance of the proposed framework, without immigrants, with different short-term memory sizes for the DTSP. Bold values indicate the best results.

$f = 100, m \Rightarrow$	0.1	0.25	0.5	0.75
Alg. & Inst.		kroA100		
$K_s = 1$	**23403.69**	25789.93	33345.29	45785.38
$K_s = 6$	23414.59	**25676.11**	**32636.93**	**44408.31**
$K_s = 10$	24151.29	26348.11	33291.71	44976.31
$K_s = 25$	26101.84	28727.52	35867.99	48225.68
Alg. & Inst.		kroA150		
$K_s = 1$	30415.70	34395.95	41691.53	57877.48
$K_s = 6$	**29870.47**	**33410.98**	**40280.44**	**55412.54**
$K_s = 10$	30735.97	34323.47	40964.49	55815.99
$K_s = 25$	33191.44	36865.50	43934.01	59251.70
Alg. & Inst.		kroA200		
$K_s = 1$	34531.48	38619.43	47531.08	67269.41
$K_s = 6$	**33624.40**	**37283.06**	**45396.54**	**64373.36**
$K_s = 10$	34710.26	38230.56	46107.69	64447.82
$K_s = 25$	37642.80	41491.16	49469.89	68129.84

On the same problem instance with $m = 0.5$ and $m = 0.75$, the values of $\mathcal{MM}$AS($\rho = 0.6$) are 32864.74 and 44994.17, respectively, whereas the corresponding values of the proposed framework with $K_s = 6$ are 32636.93 and 44408.31, respectively, which means that the latter one performs better than the former one for both dynamic test cases. However, on the remaining problems instances, i.e., kroA150 and kroA200 the proposed framework with $K_s = 6$ performs better than $\mathcal{MM}$AS($\rho = 0.6$) in all dynamic cases. For example, in kroA200 with $m = 0.1$, the performance of the former one is 33624.40 and the performance of the latter one is 33741.35.

In general, as the problem size and the magnitude of change increase the performance of $\mathcal{MM}$AS is degraded, which supports our claim in Section 6.5 that "the time required to adapt to the new environment may depend on the problem size and the degree of change". This is because if the environmental change is significant then it will take longer to eliminate unused pheromone trails. The proposed framework overcomes this problem since it removes the previous pheromone trails directly, similarly as in P-ACO.

8.3.2.4 Effect of Long-Term Memory Size K_l

For the results shown in Table 8.3 and Figure 8.2, MIACO with different K_l sizes is applied to cyclic environments with different number of cyclic states. The replacement rate for memory-based immigrants is set to $r = 0.3$. The population size of MIACO depends on the size of K_l since the ants stored in k_{long} are used as detectors, and they are re-evaluated to detect an environmental change. Therefore, the number of ants in the population of MIACO is set as $\mu' = \mu - K_l$. Since MIACO is developed to address dynamic environments where previously visited environments re-appear, an ideal K_l value is to be equal with the number of states in the cyclic environment. More precisely, each solution stored in k_{long} will represent one environment that can be used when the specific environment re-appears.

It can be observed that when $K_l = 1$, MIACO performs better when the environment changes slightly, i.e., when $m = 0.1$ and $m = 0.25$, whereas for a value $2 \leq K_l \leq 4$ MIACO performs better when the environment changes significantly, i.e., when $m = 0.5$ and $m = 0.75$. This is because the environments are more likely to be similar and there is no need to store several solutions in k_{long}.

Table 8.3: Offline performance of the proposed framework with different long-term memory size for MIACO on cyclic changing DTSPs. Bold values indicate the best results.

$f = 50, m \Rightarrow$	0.1	0.25	0.5	0.75
Alg. & Inst.		kroA100(4 cyclic states)		
MIACO($K_l = 1$)	**23313.50**	**26013.27**	32663.93	45448.51
MIACO($K_l = 2$)	23360.53	26344.74	31268.99	39287.60
MIACO($K_l = 4$)	23333.32	26372.99	**31217.16**	**39205.91**
MIACO($K_l = 6$)	23386.83	26390.08	31338.78	39367.70
MIACO($K_l = 10$)	23563.70	26487.74	31464.35	39455.40
Alg. & Inst.		kroA100(6 cyclic states)		
MIACO($K_l = 1$)	**23480.31**	**25993.23**	32634.60	45446.53
MIACO($K_l = 2$)	23569.25	25999.49	31732.32	39844.28
MIACO($K_l = 4$)	23525.57	25935.75	**31602.30**	**39709.84**
MIACO($K_l = 6$)	23644.43	26020.93	31674.05	39923.23
MIACO($K_l = 10$)	23694.87	26126.08	31877.01	40037.39
Alg. & Inst.		kroA100(10 cyclic states)		
MIACO($K_l = 1$)	**23503.41**	**25994.19**	32731.99	45439.68
MIACO($K_l = 2$)	23581.77	26185.05	**31248.77**	40562.32
MIACO($K_l = 4$)	23522.15	26158.51	31280.45	**40466.86**
MIACO($K_l = 6$)	23603.22	26202.38	31353.91	40632.69
MIACO($K_l = 10$)	23674.62	26340.72	31460.02	40805.63

Furthermore, the size of K_l is not dependent on the number of states of the cyclic environment which can be observed from the three problems with different number of cyclic states. For example, in kroA100(4 cyclic states) the performance of MIACO($K_l = 4$) is similar in kroA100(10 cyclic states). This is because one solution in k_{long} may be useful for several environments. Therefore, there is no need to store one solution for each cyclic state to represent one environment. In this way,

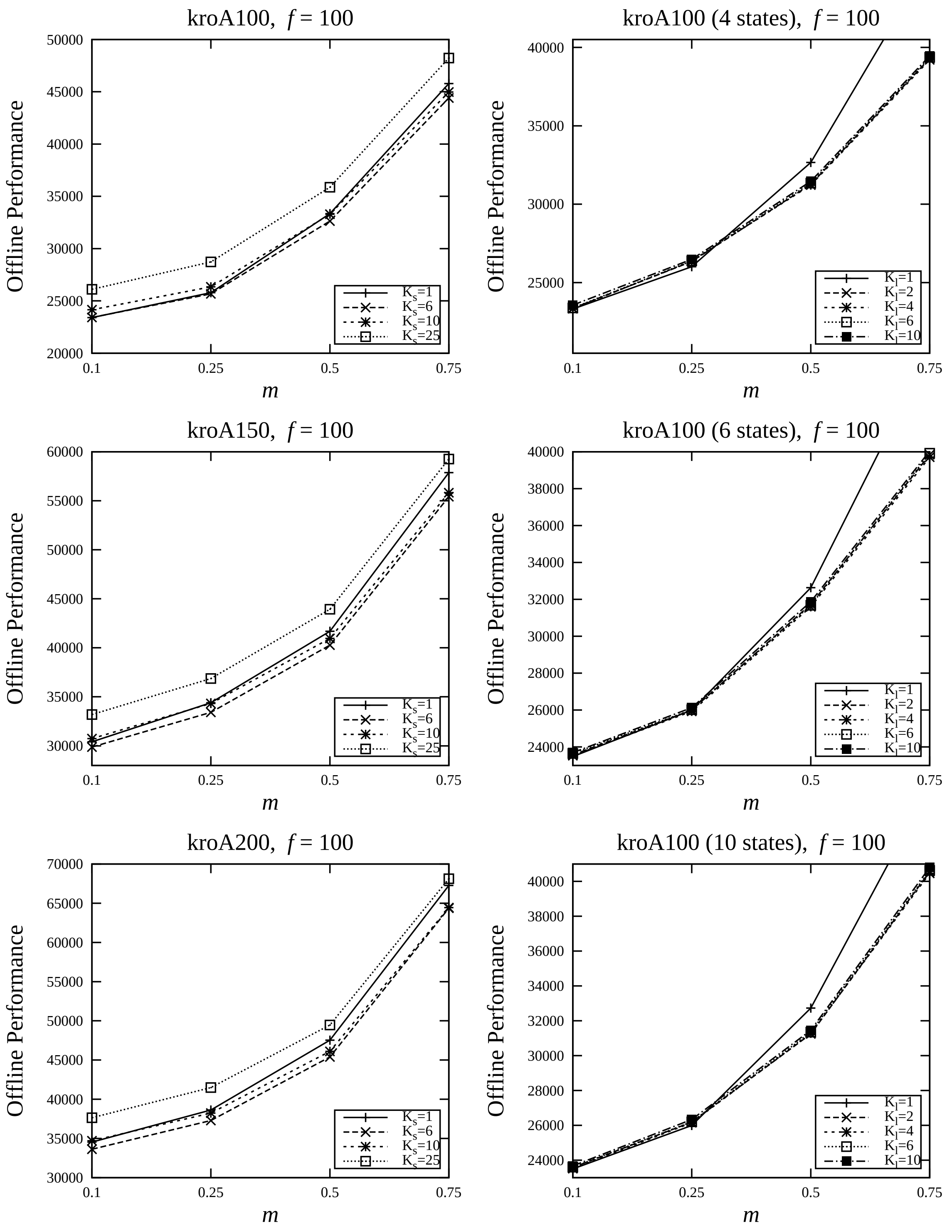


FIGURE 8.2: Offline performance of varying the size of the memory structures, i.e., short-term memory (left) and long-term memory (right), on different DTSPs.

function evaluations are not wasted.

8.3.2.5 Performance Analysis in Random Environments

The experimental results regarding the offline performance of the proposed algorithms in DTSPs with random traffic factors are presented in Table 8.4. The corresponding statistical results are presented in Table 8.5. Moreover, to better understand the dynamic behaviour of algorithms, the offline performance for the first 20 environments (100 iterations/5) is plotted in Figure 8.3 for $f = 5$ and $m = 0.1$ and $m = 0.75$, respectively, and the offline performance for the first 10 environments (500 iterations/50) is plotted in Figure 8.4 for $f = 50$ and $m = 0.1$ and $m = 0.75$, respectively. From the experimental results, several observations can be made by comparing the behaviour of the algorithms.

First, RIACO outperforms EIACO and MIACO in almost all dynamic cases with $f = 5$ and $m = 0.75$; see the comparisons of RIACO $\Leftrightarrow$ EIACO and RIACO $\Leftrightarrow$ MIACO in Table 8.5. This is because both EIACO and MIACO use knowledge based on ants from previous environments to generate immigrant ants, and thus, when not enough time is available to converge to a good solution, it is difficult to transfer useful knowledge. RIACO generates diversity randomly that may be more suitable on dynamic cases with $m = 0.75$, where the changing environments are not similar. On the other hand, when the magnitude of change is small, i.e., $m = 0.1$, even when $f = 5$, EIACO and MIACO outperform RIACO. This is because random immigrants may reach high levels of randomization and disturb the optimization process.

Second, EIACO outperforms RIACO in all dynamic cases with $f = 50$ and $f = 100$; see the comparisons of RIACO $\Leftrightarrow$ EIACO in Table 8.5. This is because the population has enough time to converge to a good optimum before a dynamic change occurs. Therefore, the knowledge transferred with the generation of elitism-based immigrants is more useful. The pheromone trails of the new environment are gene-

Table 8.4: Offline performance of the proposed algorithms with immigrants in random DTSPs.

Alg. & Inst.	pr76				pr152				pr299			
$f = 5, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO	132975	152034	189087	260507	92035	103511	126235	169585	66199	76346	97020	135038
EIACO	131478	151542	189108	260965	90598	102793	126248	170030	65143	75625	96764	135260
MIACO	131621	151300	189326	260801	90367	102799	126207	170045	65122	75619	96818	135210
$f = 50, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO	125344	137969	171101	223288	84665	93198	111715	147320	60568	67649	83671	114974
EIACO	122176	136126	169398	221994	83608	92234	110468	145418	59595	66881	82734	113471
MIACO	122144	135960	169562	221828	83398	92339	110422	145570	59649	66817	82644	113570
$f = 100, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO	123111	136769	166829	218761	82857	90713	108788	141367	58988	65671	80564	112862
EIACO	121026	134773	164601	217610	81877	89674	107167	139293	58062	64607	79611	110482
MIACO	121157	134865	164626	218009	81894	89788	107217	139460	57948	64662	79565	110901

Table 8.5: Statistical test results regarding the offline performance of the algorithms in random DTSPs. "$s+$" or "$s-$" indicates that the first algorithm is significantly better or the second algorithm is significantly better, respectively, and "+" or "−" indicates that the first algorithm is insignificantly better or the second algorithm is insignificantly better, respectively.

Alg. & Inst.	pr76				pr152				pr299			
$f = 5$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	$s-$	$s-$	$+$	$s+$	$s-$	$s-$	$+$	$s+$	$s-$	$s-$	$s+$	$s+$
RIACO⇔MIACO	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
EIACO⇔MIACO	$+$	$s-$	$s+$	$-$	$s-$	$-$	$+$	$-$	$+$	$+$	$-$	$+$
$f = 50$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
RIACO⇔MIACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
EIACO⇔MIACO	$s+$	$+$	$-$	$+$	$-$	$-$	$-$	$s+$	$+$	$-$	$-$	$+$
$f = 100$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
RIACO⇔MIACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
EIACO⇔MIACO	$+$	$+$	$+$	$+$	$+$	$+$	$+$	$+$	$-$	$+$	$-$	$s+$

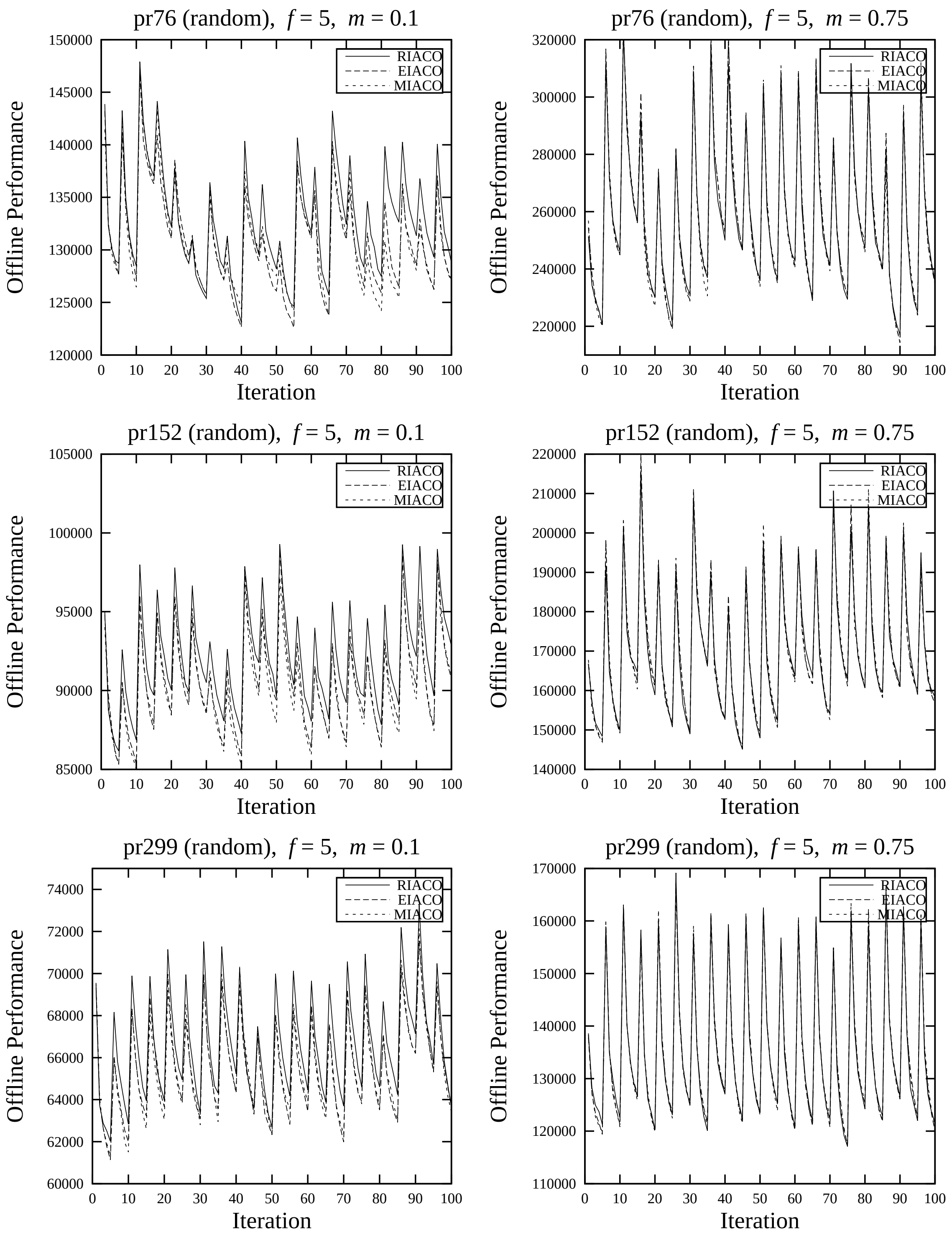


Figure 8.3: Dynamic behaviour of the algorithms for random DTSPs with fast changing environments.

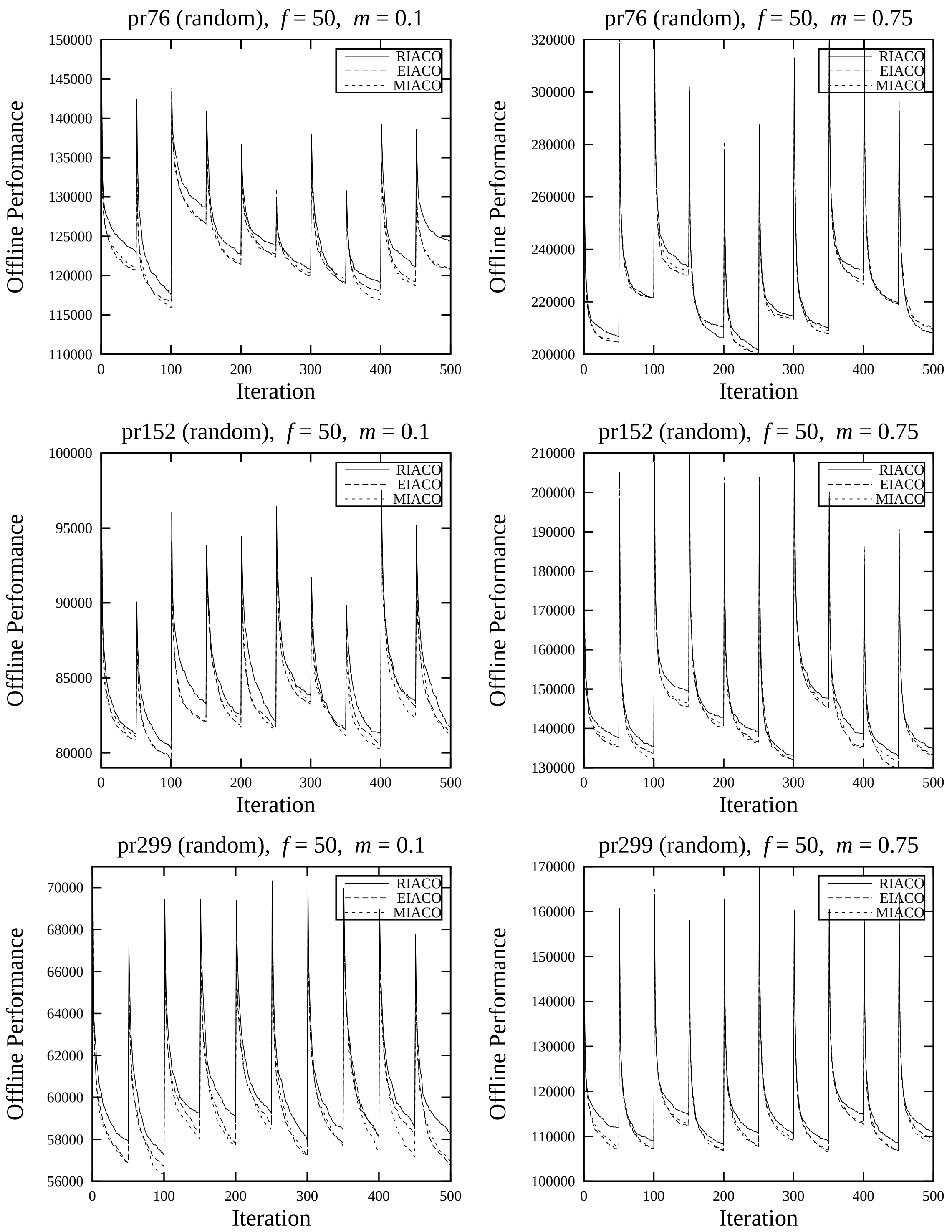


Figure 8.4: Dynamic behaviour of the algorithms for random DTSPs with slowly changing environments.

rated to promising areas, which speed up optimization. When $f = 100$, EIACO outperforms, either significantly or insignificantly, MIACO in almost all dynamic cases, since the elitism mechanism has enough time between the two environments to express its effect.

Third, MIACO has a similar behaviour with EIACO when compared with RIACO. This is because MIACO is a generalization of the elitism mechanism of EIACO. More precisely, EIACO and MIACO are similar since both use guided immigrants, but MIACO is able to choose the best solution among several previous environments from the memory to generate immigrants when the environment changes, whereas EIACO choose the best solution from the previous environment since the last environmental change to generate immigrants when the environment changes. This can be observed when $f = 50$ where the two algorithms are comparable; see the comparisons EIACO $\Leftrightarrow$ MIACO in Table 8.5.

8.3.2.6 Performance Analysis in Cyclic Environments

The experimental results regarding the offline performance of the proposed algorithms in DTSPs with cyclic traffic factors are presented in Table 8.6. The corresponding statistical results are presented in Table 8.7. Moreover, to better understand the dynamic behaviour of algorithms, the offline performance for the first 20 environments (100 iterations/5) is plotted in Figure 8.5 for $f = 5$ and $m = 0.1$ and $m = 0.75$, and the offline performance for the first 10 environments (500 iterations/50) is plotted in Figure 8.6 for $f = 50$ and $m = 0.1$ and $m = 0.75$. From the experimental results, several observations can be made by comparing the behaviour of the algorithms.

First, the performance of RIACO in cyclic DTSPs matches the performance in random DTSPs (except in pr152); see the comparisons of RIACO $\Leftrightarrow$ EIACO in Table

Table 8.6: Offline performance of the proposed algorithms with immigrants in cyclic DTSPs.

Alg. & Inst.	pr76				pr152				pr299			
$f = 5, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO	129910	145530	174108	209318	87994	99940	122442	144714	64042	72319	91831	114474
EIACO	128229	144515	174380	209766	84842	99127	122046	144451	62480	70805	91697	114503
MIACO	128031	144523	174110	209700	84481	98990	121915	144275	62489	70772	91520	114335
$f = 50, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO	123255	134213	154301	186060	82973	90139	108914	127253	58988	65429	79761	97809
EIACO	121767	132704	153530	184697	81517	89785	107833	126122	58032	64423	79043	97217
MIACO	121505	132233	153494	184174	81537	89418	107813	125948	57806	64303	78928	97093
$f = 100, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO	122221	132647	151599	180891	81457	88240	106108	123931	57953	63661	76998	94441
EIACO	120197	130406	150256	179176	80418	87708	104969	122678	56778	62592	75936	93519
MIACO	120512	130296	150286	179192	80434	87488	105000	122308	56675	62284	75832	93430

TABLE 8.7: Statistical test results regarding the offline performance of the algorithms in cyclic DTSPs. "$s+$" or "$s-$" indicates that the first algorithm is significantly better or the second algorithm is significantly better, respectively, and "+" or "−" indicates that the first algorithm is insignificantly better or the second algorithm is insignificantly better, respectively.

Alg. & Inst.	pr76				pr152				pr299			
$f = 5$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	$s-$	$s-$	$+$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$+$
RIACO⇔MIACO	$s-$	$s-$	$+$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
EIACO⇔MIACO	$+$	$+$	$s-$	$+$	$s-$	$s-$	$s-$	$s-$	$-$	$+$	$s-$	$s-$
$f = 50$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
RIACO⇔MIACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
EIACO⇔MIACO	$-$	$s-$	$-$	$s-$	$+$	$s-$	$-$	$-$	$s-$	$-$	$-$	$-$
$f = 100$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
RIACO⇔MIACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
EIACO⇔MIACO	$+$	$-$	$+$	$+$	$+$	$-$	$+$	$s-$	$-$	$s-$	$-$	$-$

8.7. However, MIACO is comparable with RIACO even in fast changing environments, since the environments re-appear several times. In some environments, i.e., $f = 5$ with $m = 0.75$ in pr76, MIACO is outperformed by RIACO. This is because when the environment changes quickly, the best memory point may not be able to store useful solutions for the current environment and hence may misguide the immigrants to non-promising areas.

Second, EIACO is outperformed by MIACO in almost all dynamic cases, either significantly or insignificantly, with $f = 5$, $f = 50$ and $f = 100$ in cyclic DTSP; see the comparisons of EIACO $\Leftrightarrow$ MIACO in Table 8.7. In comparison with the results of random DTSPs in Table 8.5, where EIACO had a slight advantage against MIACO, the performance of EIACO is inferior over that of MIACO in cyclic DTSPs. The reason is that, even if the two algorithms are similar, MIACO has an advantage against EIACO when previously visited environments appear again in the future. This can be observed in Figure 8.6 where MIACO is able to maintain better performance in the dynamic environment. On the contrast, the performance of EIACO is superior over that of RIACO even in cyclic DTSP, in which similar observations were found in the random DTSPs before. Third, MIACO outperforms EIACO and RIACO in almost all dynamic cases, either significantly or insignificantly, with $f = 50$ and $f = 100$ in cyclic DTSPs; see the comparisons of RIACO $\Leftrightarrow$ MIACO and EIACO $\Leftrightarrow$ MIACO in Table 8.7. As mentioned above, RIACO may disturb the optimization process due to the high randomization it generates. The reason why MIACO performs better in cyclic DTSPs than in random DTSPs is that it can move the population directly to any previously visited environment. MIACO stores the best solutions for all cyclic base states and reuses them by generating memory-based immigrants, whereas EIACO moves the population to an area which is similar with the previous environment only. As the environments cycle (re-appear) more times, more knowledge is gained to the memory in MIACO, and memory expresses

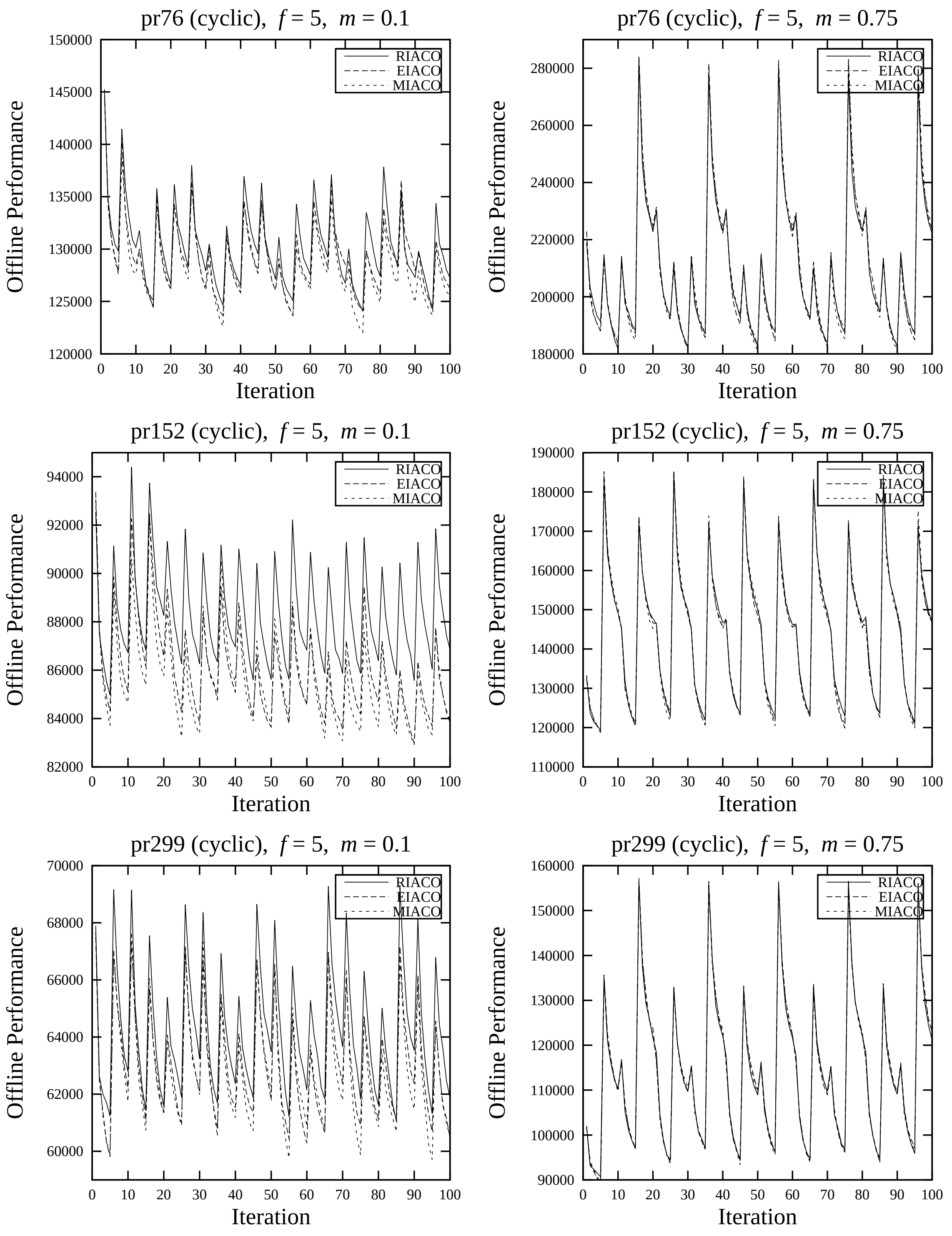


Figure 8.5: Dynamic behaviour of the algorithms for cyclic DTSPs with fast changing environments.

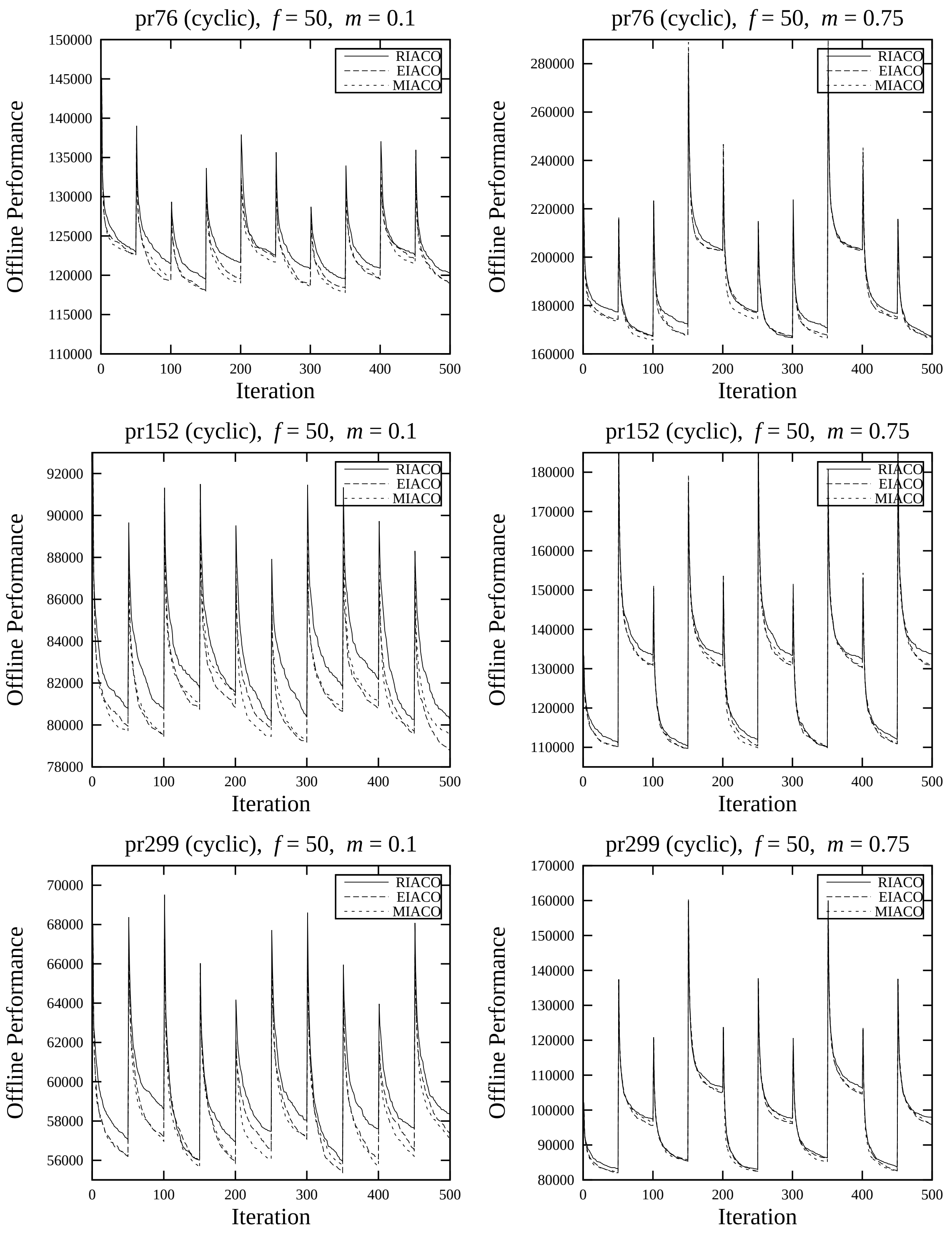


Figure 8.6: Dynamic behaviour of the algorithms for cyclic DTSPs with slowly changing environments.

Table 8.8: Offline performance of the proposed framework with different replacement rates of random immigrants on fast changing DTSPs. Bold values indicate the best results.

$f = 5, m \Rightarrow$	0.1	0.25	0.5	0.75
Alg. & Inst.		pr76		
RIACO($r = 0.0$)	**132195**	**151766**	**188893**	260958
RIACO($r = 0.1$)	132805	151800	189041	**260340**
RIACO($r = 0.3$)	132975	152034	189087	260507
RIACO($r = 0.5$)	134751	153213	190194	262004
RIACO($r = 0.8$)	139942	157912	195374	269299
Alg. & Inst.		pr152		
RIACO($r = 0.0$)	**90824**	**102559**	**125618**	170123
RIACO($r = 0.1$)	91553	103326	126028	169840
RIACO($r = 0.3$)	92035	103511	126235	**169585**
RIACO($r = 0.5$)	92849	103844	126782	170807
RIACO($r = 0.8$)	95603	105866	129136	174453
Alg. & Inst.		pr299		
RIACO($r = 0.0$)	**65131**	**75316**	**96265**	136108
RIACO($r = 0.1$)	65717	75902	96650	135530
RIACO($r = 0.3$)	66199	76346	97020	**135038**
RIACO($r = 0.5$)	67223	77436	97994	136040
RIACO($r = 0.8$)	69584	79738	100469	139286

its effect.

8.3.2.7 Effect of Immigrants Replacement Rate r

In order to investigate the effectiveness of the immigrants replacement rate, further experiments were performed on the same problem instances with the same

Table 8.9: Offline performance of the proposed framework with different replacement rates of elitism-based immigrants on slowly changing DTSPs. Bold values indicate the best results.

$f = 50, m \Rightarrow$	0.1	0.25	0.5	0.75
Alg. & Inst.		pr76		
EIACO($r = 0.0$)	124827	137938	171135	222960
EIACO($r = 0.1$)	123042	136844	170214	**222263**
EIACO($r = 0.3$)	122176	**136126**	**169398**	223288
EIACO($r = 0.5$)	**121123**	136184	170351	225151
EIACO($r = 0.8$)	121180	137550	173809	231944
Alg. & Inst.		pr152		
EIACO($r = 0.0$)	84387	92858	111201	146613
EIACO($r = 0.1$)	83841	92421	110598	146006
EIACO($r = 0.3$)	83608	**92234**	**110468**	**145418**
EIACO($r = 0.5$)	**83477**	92697	110781	145919
EIACO($r = 0.8$)	84486	94447	113972	150618
Alg. & Inst.		pr299		
EIACO($r = 0.0$)	60313	67265	83244	114246
EIACO($r = 0.1$)	59775	66942	82976	113776
EIACO($r = 0.3$)	59595	**66881**	**82734**	**113471**
EIACO($r = 0.5$)	**59516**	67381	83594	114705
EIACO($r = 0.8$)	60853	70182	88275	121821

parameters used before but with different immigrant replacement rates, i.e., $r \in \{0.0, 0.1, 0.3, 0.5, 0.8\}$, where $r = 0.0$ means that no immigrants are generated to replace ants in the $k_{short}(t)$. RIACO and EIACO (or MIACO) are applied on the dynamic cases that showed good performance from the basic experiments above, which make use of random and guided immigrants, respectively.

In Table 8.8 and Figure 8.7, RIACO is applied in fast changing environments, which

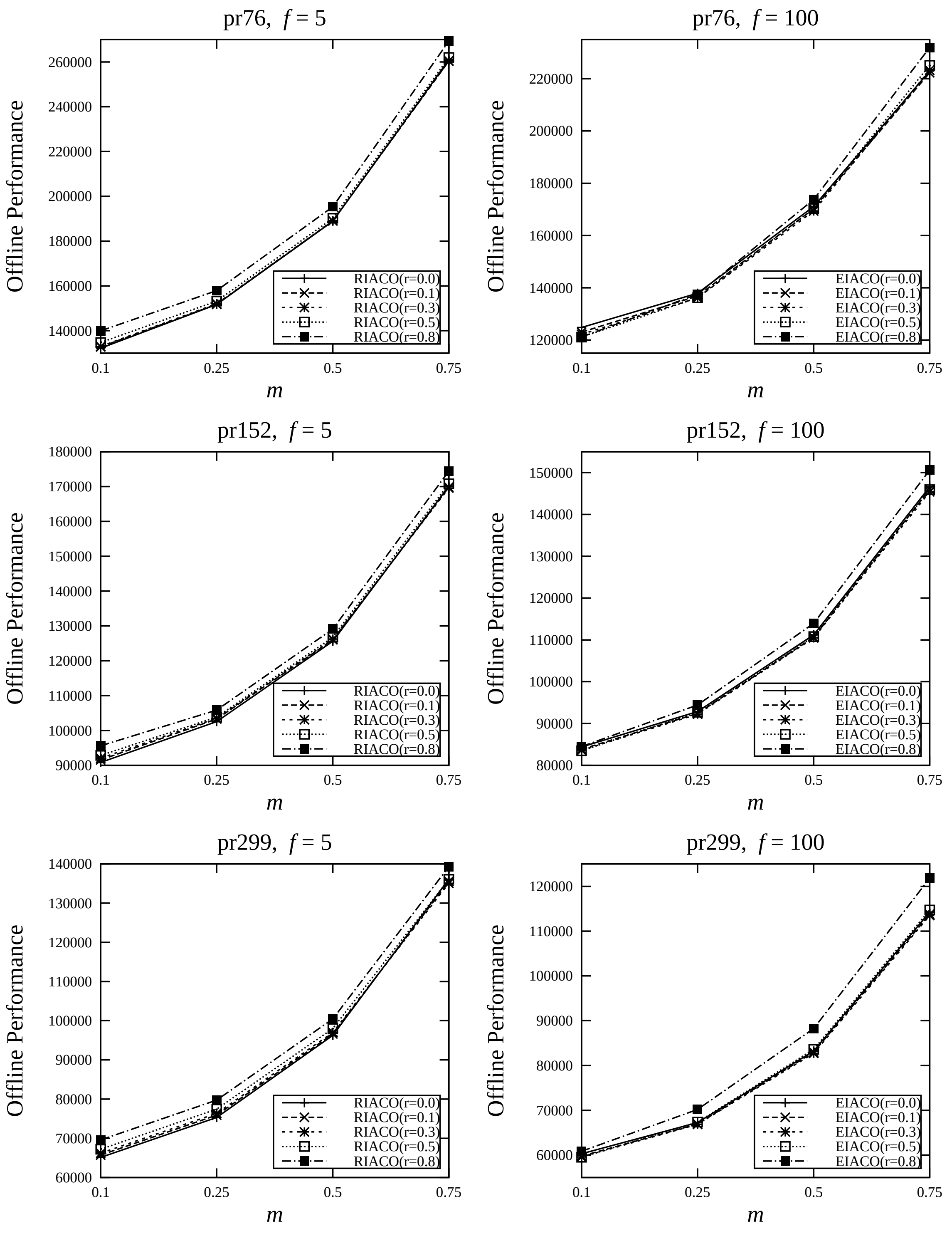


FIGURE 8.7: Offline performance of varying the replacement rate of immigrants, i.e., random immigrants (left) and guided immigrants (right), on different DTSPs.

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

In Table 8.9 and Figure 8.7, EIACO is applied in slowly changing environments, which shows that guided immigrants improve the performance of ACO in all cases when the replacement rate is $r = 0.3$, and confirms that EIACO performs well on slowly changing environments. Furthermore, when the replacement rate is $r > 0.3$, the performance is usually degraded and sometimes is even worse than EIACO($r = 0.0$). This shows that too much knowledge transferred may start the run in a new environment from a local optimum (or a near local optimum) location and get the algorithm stuck there, which validates our expectation mentioned before.

8.3.2.8 Comparison with other Peer ACO Algorithms in DTSP

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

We compare the proposed algorithms with other peer ACO algorithms, i.e., $\mathcal{MM}$AS described in Section 4.6.1 and P-ACO, SC-PACO, FS-PACO and M-ACO described

TABLE 8.10: Experimental results regarding offline performance on the DTSP with $m = rand[0,1]$ and $f = rand[1,100]$ in att532. "$s+$" or "$s-$" indicates that the algorithm in the row is significantly better than the one in the column or the opposite and "$\sim$" indicates no significance.

Alg.	RIACO	EIACO	MIACO	$\mathcal{MM}$AS	P-ACO	SC-PACO	FS-PACO	M-ACO
Offl. Perf.	147646	145968	146240	150436	156287	152848	150185	155176
Std. Dev.	523.14	507.91	585.91	292.57	691.61	756.33	643.04	503.27
				Wilcoxon Sum-Rank Test				
RIACO		$s-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$
EIACO	$s+$		$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIACO	$s+$	$s-$		$s+$	$s+$	$s+$	$s+$	$s+$
$\mathcal{MM}$AS	$s-$	$s-$	$s-$		$s+$	$s+$	$\sim$	$s+$
P-ACO	$s-$	$s-$	$s-$	$s-$		$s-$	$s-$	$s-$
SC-PACO	$s-$	$s-$	$s-$	$s-$	$s+$		$s-$	$s+$
FS-PACO	$s-$	$s-$	$s-$	$\sim$	$s+$	$s+$		$s+$
M-ACO	$s-$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	

in Section 6.6.1. Note that since the time interval of such kind of environment varies, many existing approaches used in ACO algorithms for DTSPs, e.g., global and local restart strategies ($\tau - strategy$ and $\eta - strategy$) or the ACO-*shaking*, also described in Section 6.6.1, cannot be applied because they do not have any mechanism to detect dynamic changes. The parameter settings for RIACO, EIACO and MIACO are the same as in the basic experiments above, whereas the rest are shown in Appendix C.

The experimental results regarding the offline performance of the aforementioned algorithms are presented in Table 8.10 with the corresponding statistical results. In Figure 8.8 the dynamic behaviour of the algorithms is presented with corresponding m values shown in Figure 8.9. From the experimental results several observations can be drawn.

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

Second, all the proposed algorithms perform significantly better than $\mathcal{MM}$AS and P-ACO. This is because $\mathcal{MM}$AS uses only pheromone evaporation to eliminate pheromone trails from the previous environment that are not useful to a new environment, and thus, needs sufficient time to adapt to the changing environments. On the other hand, P-ACO eliminates pheromone trails directly if an ant is removed from $k_{long}(t)$. However, if identical ants are stored in the $k_{long}(t)$, then the algorithm will reach stagnation behaviour, and thus, needs sufficient time to escape from it.

Third, FS-PACO and SC-PACO perform significantly better than P-ACO, whereas the former algorithm performs significantly better than the latter algorithm. Both algorithms are variations of the P-ACO algorithm that aims to increase and maintain diversity in $k_{long}(t)$. Hence, SC-PACO and FS-PACO adapt better to the changing

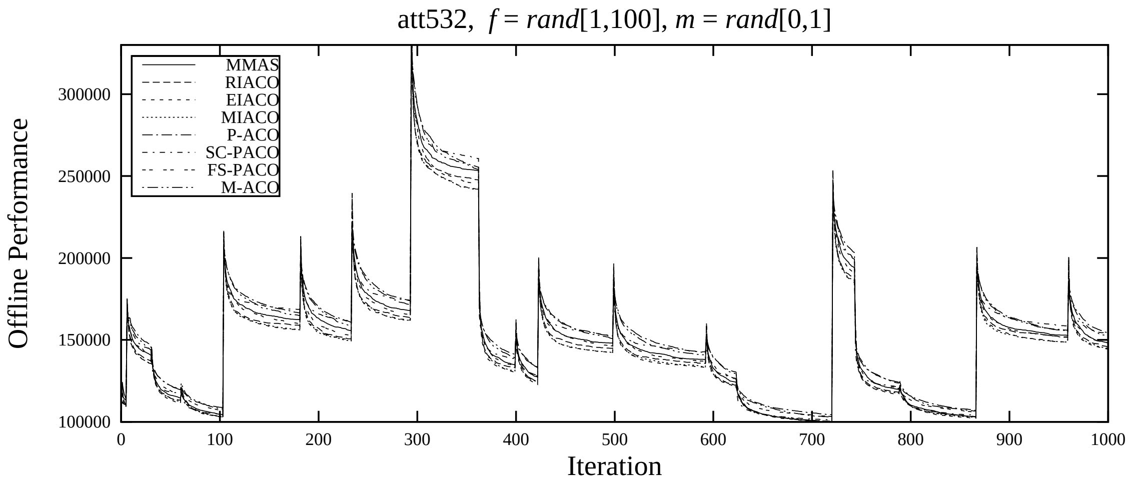


FIGURE 8.8: Dynamic behaviour of the proposed algorithms in comparison with other peer ACO algorithms on the DTSP with $m = rand[0,1]$ and $f = rand[1,100]$.

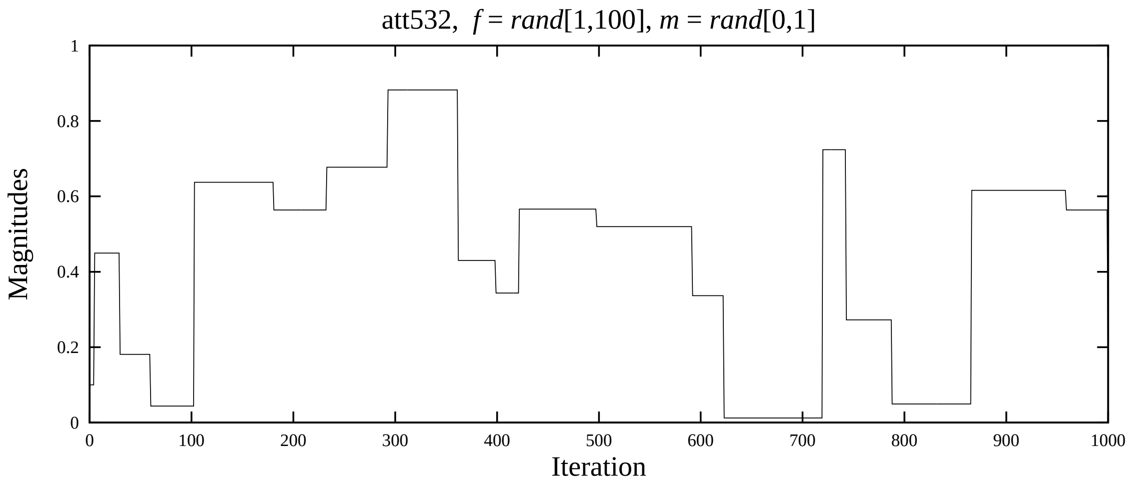


FIGURE 8.9: Varying values for $m = rand[0,1]$ and $f = rand[1,100]$ used for the DTSP.

environment than P-ACO probably because the simple crowding policy and the fitness sharing used in the algorithms, respectively, address the stagnation behaviour in P-ACO.

Fourth, M-ACO is significantly better than P-ACO since the local search that is integrated promotes exploitation to improve the solution quality, and the risk of

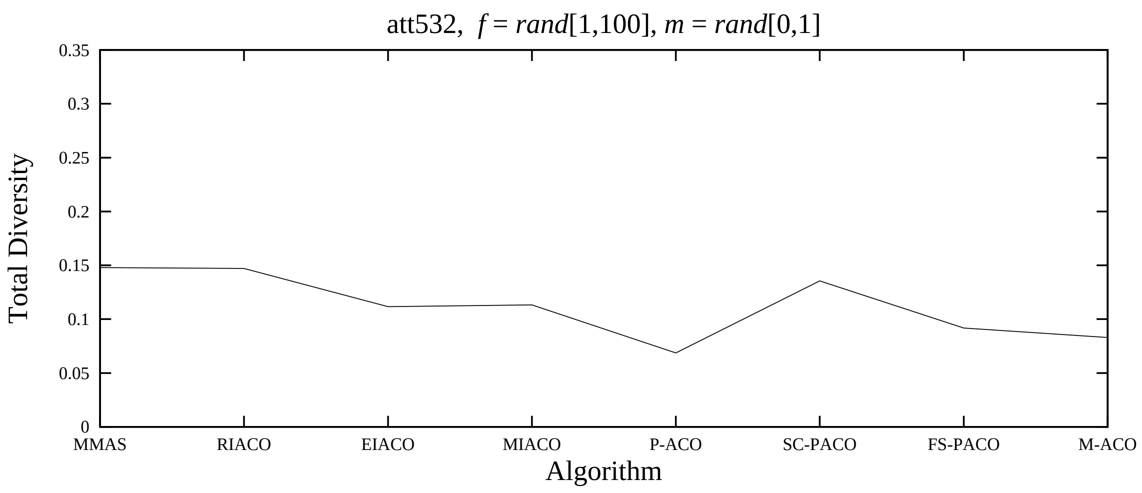


Figure 8.10: Total diversity of the proposed ACO algorithms in comparison with other peer ACO algorithms on the DTSP with $m = rand[0,1]$ and $f = rand[1,100]$.

stagnation behaviour is eliminated using a diversity increase scheme based on traditional immigrants. Whenever $k_{long}(t)$ contains identical ants, a random immigrant replaces an ant until the algorithm generates sufficient diversity.

Finally, in order to investigate the effect of the diversity generated, we calculate the total diversity using Equation 6.7. The total diversity results for the algorithms are presented in Figure 8.10. It can be observed that $\mathcal{MM}$AS has a higher diversity than most algorithms. The P-ACO algorithm has the lowest diversity level, which shows the effect when identical ants are stored in the population-list. RIACO maintains the highest diversity among the algorithms with immigrants schemes, since diversity is generated randomly, whereas EIACO and MIACO generate guided diversity via transferring knowledge. FS-PACO, SC-PACO and M-ACO increase the diversity of P-ACO with the different strategies used, but SC-PACO maintains higher levels of diversity. Considering the results of the total diversity with the those of the offline performance in Table 8.10 shows that ACO algorithms that maintain high diversity levels do not always achieve better performance than other ACO algorithms for the DTSP.

8.3.2.9 Summary for the DTSP Experimental Results

In this subsection we summarize all the results from the experiments of DTSP. In Figure 8.11 the results from Table 8.5 and Table 8.7 are transformed to squares, that indicate the relative performance of the algorithms. The vertical and horizontal axes indicate the magnitude and frequency values, respectively. The brightness of squares indicate how well the algorithm performs on a specific dynamic test case. For example, if an algorithm is outperformed from all its competitors in all problem instances, then the square will be white; otherwise the square will be grey. The darkness of the square depends on the number of algorithms a specific algorithm outperforms, and on the consistency in different problem instances.

In Table 8.11 the analysis of the algorithms from the experiments in DTSP is summarized, where slow and fast indicate the frequency of the dynamic environment, and where low, medium and high indicate the magnitude of the dynamic environment.

From Figure 8.11 it can be observed that:

1. RIACO performs well only in dynamic cases where $f = 5$ with $m = 0.5$ and $m = 0.75$ because the squares are darker in these cases and white on the remaining. In cyclic DTSPs the squares are brighter because the performance of RIACO is not consistent in all problem instances, e.g., is outperformed in one problem instance.

2. EIACO performs in dynamic cases where $f = 50$ and $f = 100$ with $m = 0.1$, $m = 0.25$, $m = 0.5$ and 0.75 because the squares are darker in these cases and lighter in the $f = 5$ cases. The squares of EIACO are brighter in cyclic TSPs when compared with the ones in random DTSPs because MIACO performs better.

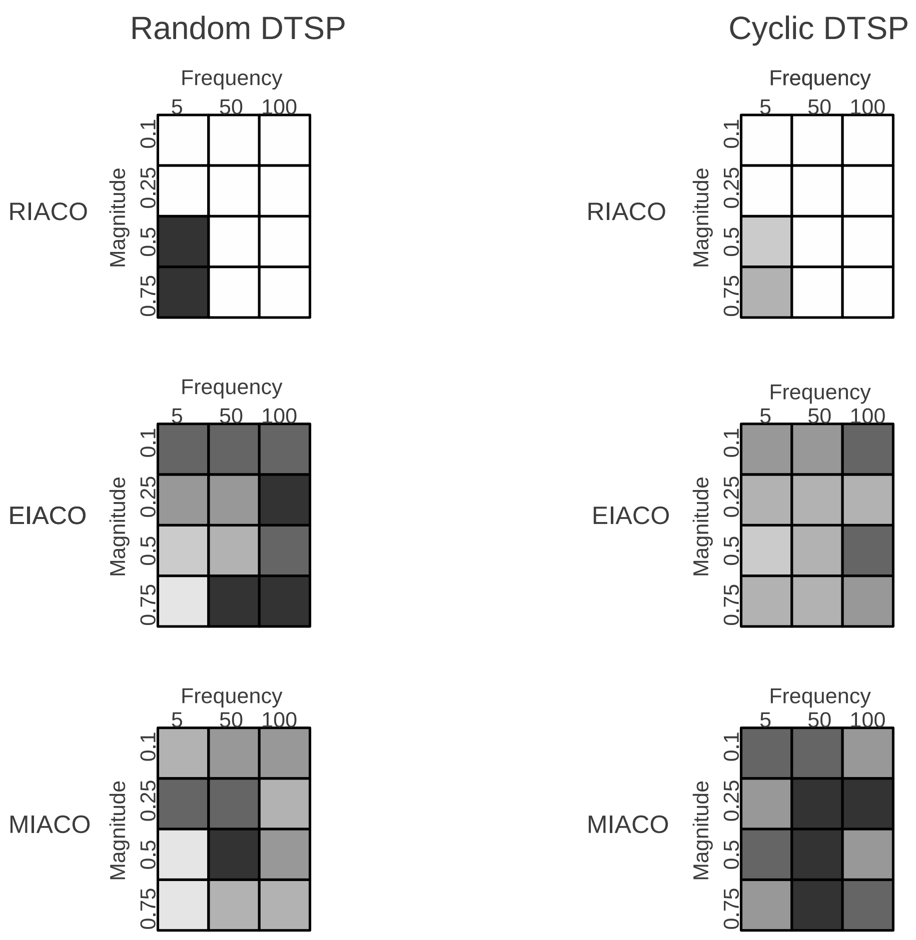


Figure 8.11: Performance of the algorithms on different dynamic test cases for the DTSP. The darkest squares indicate better performance of the algorithm when compared with its competitors considering all the problem instances.

3. MIACO performs well in random DTSPs and has similar performance with EIACO. However, EIACO performance is superior from MIACO performance because the squares of MIACO are lighter in random DTSPs. On the other hand, MIACO improves its performance in cyclic DTSPs because the squares are darker from the ones in random DTSPs. Moreover, MIACO performance is superior from EIACO performance because the squares of MIACO are darker in cyclic DTSPs.

Table 8.11: Summary observations regarding the relative performance of the algorithms in different dynamic test cases of DTSPs.

	Environmental Dynamic Case				
Algorithms	Slow	Fast	Low	Medium	High
RIACO	Disturbs the optimization process because of too much randomization.	Performs well only when the changes are severe.	Similar performance as in the “slow” case.	Similar performance as in the “slow” case.	Similar performance as in the “slow” case.
EIACO	Performs well since it has enough time to gain knowledge from previous environments.	Performs well only when the environment changes slightly.	Performs well since the knowledge transferred fits between similar environments.	Similar performance as in the “low” case.	Performs well only when the environment changes slowly.
MIACO	Similar performance with EIACO in random environments but better performance in cyclic environments because of memory usage.	Memory cannot keep track of the best solutions.	Performs better on most cyclic environments.	Similar performance as in the ”low” case.	Similar performance as in the ”low”.

8.3.3 Experimental Results for the DVRP

8.3.3.1 Parameter Settings

The parameter settings for RIACO, EIACO and MIACO in the DVRP are the same with the parameters in the DTSP. Appendix C presents all the parameters used for the DVRP. The important algorithmic parameters for ACS-DVRP, which follows the traditional ACO framework, have been optimized via experiments as presented in the next subsection.

8.3.3.2 Effect of the Evaporation Rate ρ

In Table 8.12, one of the best performing traditional ACO algorithms, i.e., ACS-DVRP, is applied to slowly changing environments with different magnitudes of change, and the evaporation rate is optimized (RIACO, EIACO and MIACO do not have pheromone evaporation). In the static VRP, the recommended evaporation rate for ACS is $\rho = 0.1$ [196].

In contrast to the observation with $\mathcal{MM}$AS in Table 8.1 for the DTSP previously, the recommended evaporation rate in the static VRP usually has good performance or competitive performance in the DVRP. This can be observed in Table 8.12 with the ACS-DVRP($\rho = 0.1$) results in F-n45-k4 and F-n72-k4. However, when ρ is high, the performance is often degraded as in $\mathcal{MM}$AS for the DTSP. When $\rho > 0.1$ the performance is sometimes improved, e.g., in all test cases of the F-n135-k7. This shows that a higher evaporation rate improves the performance in larger problem instances, e.g., F-n135-k7, and the performance is furthermore improved when the magnitude of change increases, e.g., when $m = 0.5$ and $m = 0.75$. A similar observation was found with $\mathcal{MM}$AS in Table 8.1 for the DTSP.

Table 8.12: Offline performance of ACS-DVRP with different evaporation rates for the DVRP. Bold values indicate the best results.

$f = 100, m \Rightarrow$	0.1	0.25	0.5	0.75
Alg. & Inst.		F-n45-k4		
ACS-DVRP($\rho = 0.1$)	**920.29**	**1134.74**	1871.18	**3291.76**
ACS-DVRP($\rho = 0.2$)	925.76	1139.36	1866.70	3311.35
ACS-DVRP($\rho = 0.4$)	928.51	1141.75	1864.71	3351.59
ACS-DVRP($\rho = 0.6$)	936.63	1160.26	**1862.90**	3323.81
Alg. & Inst.		F-n72-k4		
ACS-DVRP($\rho = 0.1$)	**313.17**	393.90	**664.33**	**1117.12**
ACS-DVRP($\rho = 0.2$)	314.45	**393.83**	666.77	1120.75
ACS-DVRP($\rho = 0.4$)	314.71	396.88	666.66	1118.43
ACS-DVRP($\rho = 0.6$)	315.22	398.26	665.13	1121.20
Alg. & Inst.		F-n135-k7		
ACS-DVRP($\rho = 0.1$)	1497.29	1866.97	2900.09	5011.19
ACS-DVRP($\rho = 0.2$)	**1491.20**	**1860.12**	2910.56	5005.90
ACS-DVRP($\rho = 0.4$)	1494.08	1865.71	**2888.78**	**4985.65**
ACS-DVRP($\rho = 0.6$)	1496.40	1864.13	2901.97	5008.14

In general, the evaporation rate does not have a significant impact on the performance of ACS-DVRP in dynamic environments with a small search space. This is natural because the difference between the standard ACS and $\mathcal{MM}$AS is that, in the former algorithm, pheromone evaporation is applied only to the trails of the best ant, whereas in the latter algorithm it is applied to all the trails.

8.3.3.3 Performance Analysis in Random Environments

The experimental results regarding the offline performance of the proposed algorithms in DVRPs with random traffic factors are presented in Table 8.13. The cor-

TABLE 8.13: Offline performance of the proposed algorithms with immigrants in random DVRPs

Alg. & Inst.	F-n45-k4				F-n72-k4				F-n135-k7			
$f = 5, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO	913.7	1137.0	1859.3	3176.4	326.9	417.4	683.6	1160.8	1562.5	2015.2	3256.9	5408.6
EIACO	914.3	1139.3	1860.4	3172.1	325.2	415.1	678.9	1153.3	1561.2	2011.9	3251.7	5407.0
MIACO	914.2	1140.1	1857.7	3170.6	325.1	415.2	679.7	1155.3	1560.9	2012.5	3250.1	5409.9
$f = 50, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO	866.6	1021.4	1662.6	2847.7	299.8	367.8	590.8	1024.8	1454.2	1751.8	2781.7	4746.1
EIACO	871.5	1026.6	1674.1	2825.8	294.8	362.7	573.2	1006.2	1441.3	1722.1	2702.3	4638.3
MIACO	871.4	1026.7	1662.2	2835.7	294.3	362.9	576.5	1002.9	1442.1	1724.6	2705.1	4626.8
$f = 100, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO	869.2	1024.6	1575.8	2914.6	294.1	356.6	585.8	980.4	1434.4	1719.3	2658.3	4648.1
EIACO	875.7	1023.0	1570.5	2790.3	288.7	349.8	572.4	959.7	1418.3	1702.2	2556.2	4512.7
MIACO	872.8	1024.1	1579.9	2791.0	288.5	349.9	567.2	964.3	1422.3	1702.1	2566.0	4486.8

TABLE 8.14: Statistical test results regarding the offline performance of the algorithms in random DVRPs. "$s+$" or "$s-$" indicates that the first algorithm is significantly better or the second algorithm is significantly better, respectively, and "+" or "−" indicates that the first algorithm is insignificantly better or the second algorithm is insignificantly better, respectively.

Alg. & Inst.	F-n45-k4				F-n72-k4				F-n135-k7			
$f = 5$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	$+$	$+$	$+$	$-$	$s-$	$s-$	$s-$	$s-$	$-$	$-$	$-$	$-$
RIACO⇔MIACO	$+$	$+$	$+$	$-$	$s-$	$s-$	$s-$	$s-$	$-$	$-$	$-$	$+$
EIACO⇔MIACO	$+$	$-$	$+$	$+$	$-$	$+$	$+$	$+$	$+$	$-$	$+$	$+$
$f = 50$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	$s+$	$s+$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
RIACO⇔MIACO	$s+$	$-$	$s+$	$-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
EIACO⇔MIACO	$+$	$-$	$-$	$+$	$-$	$+$	$+$	$-$	$+$	$+$	$+$	$-$
$f = 100$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	$s+$	$-$	$-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
RIACO⇔MIACO	$s+$	$-$	$+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
EIACO⇔MIACO	$-$	$+$	$+$	$+$	$-$	$+$	$-$	$-$	$+$	$-$	$+$	$-$

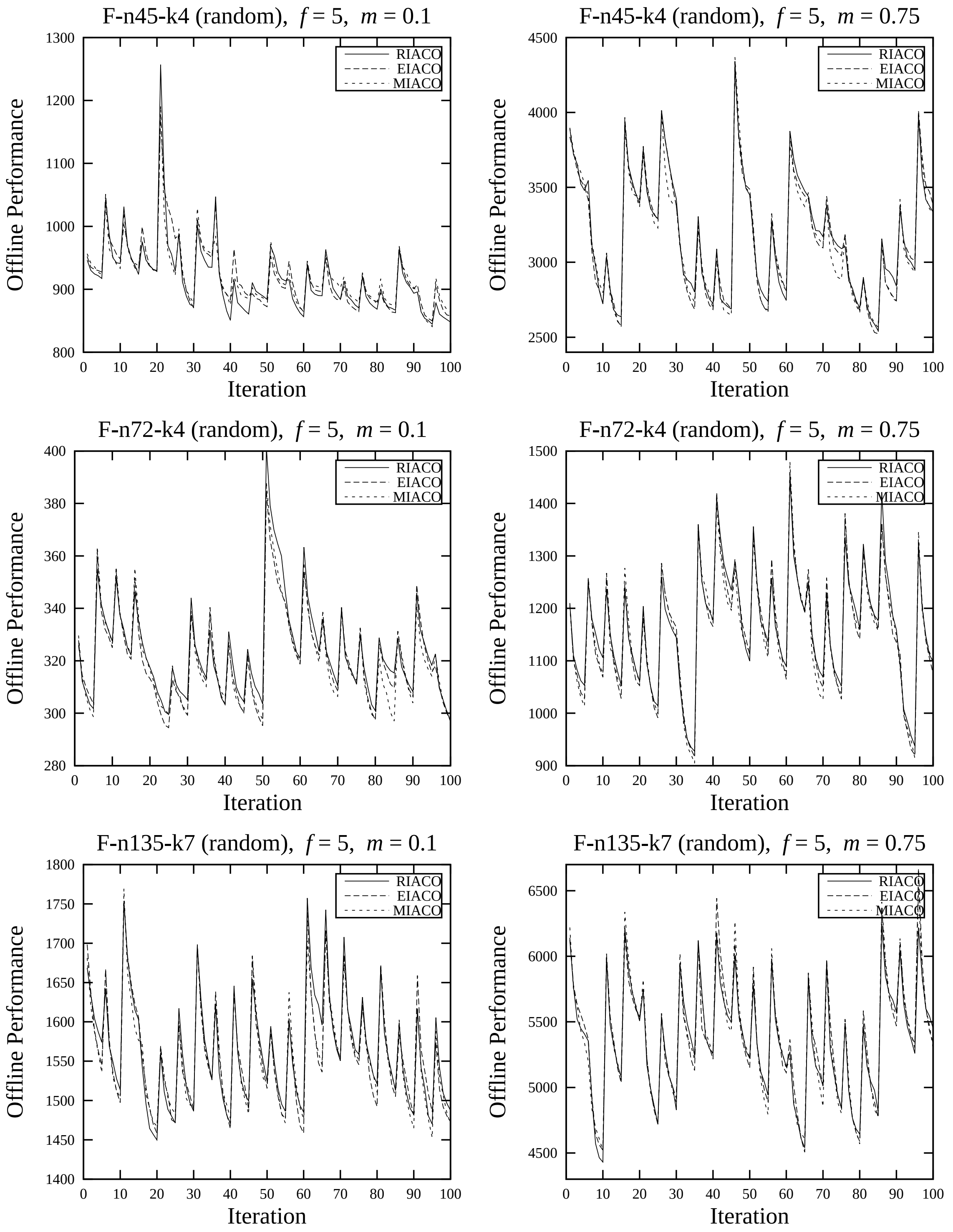


Figure 8.12: Dynamic behaviour of the algorithms for random DVRPs with fast changing environments.

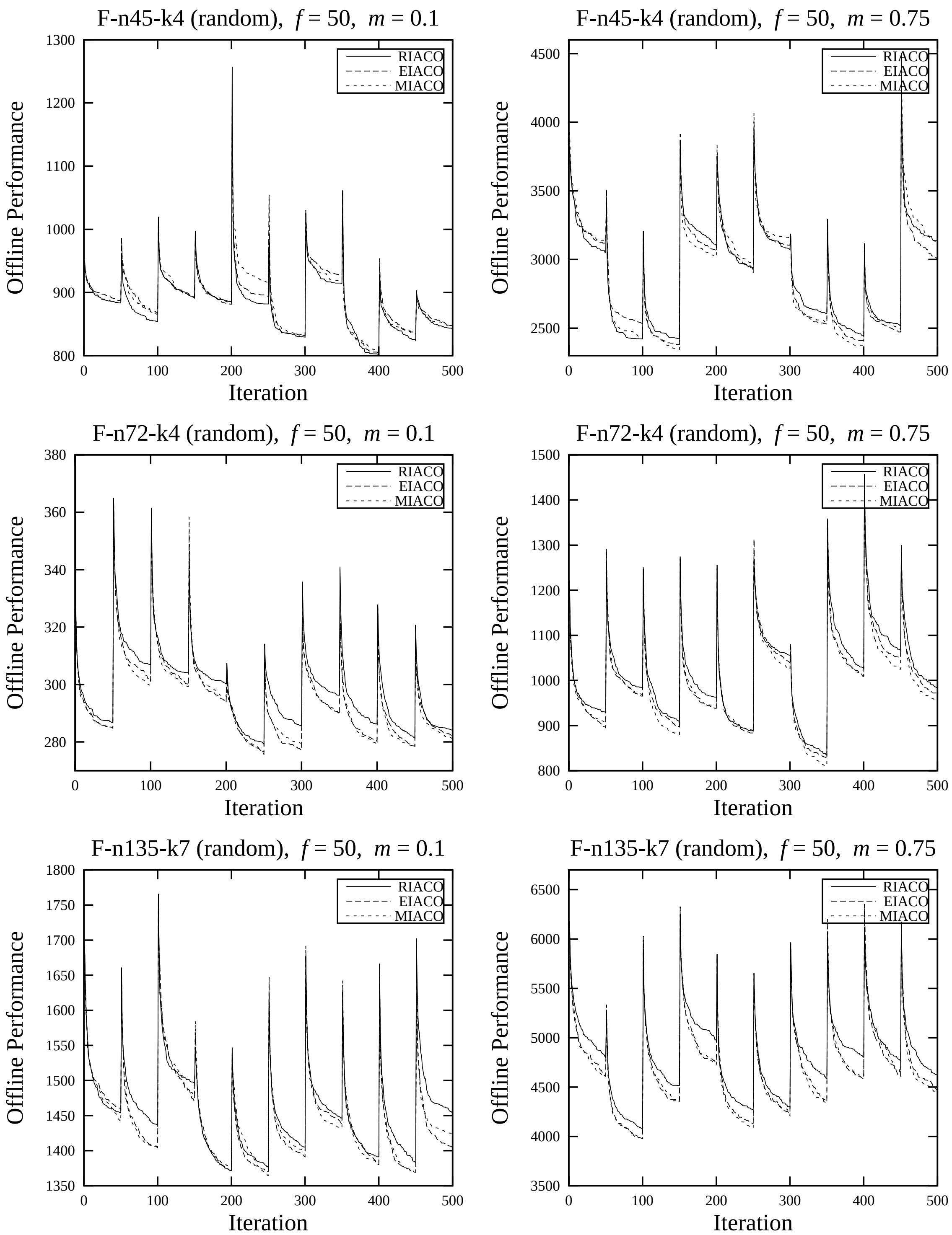


FIGURE 8.13: Dynamic behaviour of the algorithms for random DVRPs with slowly changing environments.

responding statistical results are presented in Table 8.14. Moreover, to better understand the dynamic behaviour of algorithms, the offline performance for the first 20 environments (100 iterations/5) is plotted in Figure 8.12 for $f = 5$ and $m = 0.1$ and $m = 0.75$, respectively, and the offline performance for the first 10 environments (500 iterations/50) is plotted in Figure 8.13 for $f = 50$ and $m = 0.1$ and $m = 0.75$, respectively. From the experimental results, several observations can be made by comparing the behaviour of the algorithms.

First, it was expected that RIACO would outperform EIACO and MIACO in fast and significantly changing environments, whereas EIACO and MIACO would outperform RIACO in slowly and slightly changing environments as it was observed in the DTSP previously. However, this is not true in the case of the DVRP because RIACO is outperformed by EIACO (and MIACO) in most changing environments (except in F-n45-k4); see the comparisons of RIACO $\Leftrightarrow$ EIACO and RIACO $\Leftrightarrow$ MIACO in Table 8.14. In problem instances where the search space is small, i.e., F-n45-k4, RIACO performs better in most dynamic test cases, even in fast or slowly changing environments, than its competitors. This behaviour may have several reasons: (1) the knowledge transferred in EIACO or MIACO is too much that it starts the run in the new environment from a local optimum as it was observed previously; and (2) random immigrants have a higher probability to hit the optimum in a smaller search space and the risk of randomization is less.

Second, EIACO and MIACO have similar performance in most dynamic test cases, since none of the algorithms is significantly better than the other; see the comparisons of EIACO $\Leftrightarrow$ MIACO in Table 8.14. This behaviour is natural since both algorithms generate guided immigrants. A similar observation has been found in some dynamic test cases in the experiments before for the DTSPs.

8.3.3.4 Performance Analysis in Cyclic Environments

The experimental results regarding the offline performance of the proposed algorithms in DVRPs with cyclic traffic factors are presented in Table 8.15. The corresponding statistical results are presented in Table 8.16. Moreover, to better understand the dynamic behaviour of algorithms, the offline performance for the first 20 environments (100 iterations/5) is plotted in Figure 8.14 for $f = 5$ and $m = 0.1$ and $m = 0.75$, respectively, and the offline performance for the first 10 environments (500 iterations/50) is plotted in Figure 8.15 for $f = 50$ and $m = 0.1$ and $m = 0.75$, respectively. From the experimental results, several observations can be made by comparing the behaviour of the algorithms.

First, RIACO has a similar performance for cyclic DVRP as for random DVRPs before, on some dynamic test cases in F-n45-k4; see the comparisons of RIACO $\Leftrightarrow$ EIACO and RIACO $\Leftrightarrow$ MIACO in Table 8.16. The results on F-n72-k4 and vrp145 for cyclic DVRPs match the results for random DVRPs, since RIACO is outperformed by its competitors in almost all dynamic cases. The performance of RIACO is inconsistent in both random and cyclic DVRPs and it requires further investigation; see Table 8.17.

Second, EIACO and MIACO have similar performance for cyclic DVRP as for the random DVRP above, except in some cases where MIACO performs better than EIACO, i.e., when $f = 5$ and $m = 0.25$ in F-n45-k4; see the comparisons of EIACO $\Leftrightarrow$ MIACO in Table 8.16. Different from the observation found in the cyclic DTSPs for MIACO, for DVRP, the memory is not advantageous when the environments reappear, which was not expected. This is probably because the memory is not able to store solutions that may represent previously visited environments. The similarity metric used in Equation 5.3 is based on the common edges and may not be suitable,

Table 8.15: Offline performance of the proposed algorithms with immigrants in cyclic DVRPs.

Alg. & Inst.	F-n45-k4				F-n72-k4				F-n135-k7			
$f = 5, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO	947.4	1018.3	1648.6	2616.5	322.2	422.5	577.9	729.9	1593.4	1881.4	2595.1	4583.9
EIACO	952.8	1017.1	1643.2	2614.5	319.9	420.9	576.5	726.9	1594.2	1880.3	2595.2	4583.0
MIACO	951.1	1014.8	1645.0	2612.1	319.6	421.3	577.4	727.3	1593.1	1876.8	2598.6	4584.2
$f = 50, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO	887.1	959.2	1434.2	2320.3	302.1	381.7	500.2	658.0	1477.6	1688.2	2307.5	4101.4
EIACO	888.8	956.1	1427.5	2309.6	296.6	373.4	487.3	646.4	1472.5	1668.0	2261.7	4023.8
MIACO	887.8	957.4	1432.2	2306.3	295.8	373.2	487.7	646.0	1472.8	1670.2	2263.7	4024.5
$f = 100, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO	871.1	943.3	1341.8	2241.9	296.7	375.2	481.1	643.8	1434.9	1644.9	2207.0	3877.6
EIACO	872.3	940.5	1350.6	2173.4	290.8	362.8	466.3	623.1	1430.2	1623.7	2158.2	3777.8
MIACO	871.2	943.1	1353.3	2164.2	290.0	362.9	467.0	623.3	1427.1	1622.4	2155.4	3784.1

Table 8.16: Statistical test results regarding the offline performance of the algorithms in cyclic DVRPs. "$s+$" or "$s-$" indicates that the first algorithm is significantly better or the second algorithm is significantly better, respectively, and "+" or "−" indicates that the first algorithm is insignificantly better or the second algorithm is insignificantly better, respectively.

Alg. & Inst.	F-n45-k4				F-n72-k4				F-n135-k7			
$f = 5$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	$s+$	$-$	$s-$	$-$	$s-$	$s-$	$s-$	$s-$	$+$	$-$	$+$	$-$
RIACO⇔MIACO	$s+$	$s-$	$-$	$-$	$s-$	$s-$	$-$	$s-$	$-$	$s-$	$+$	$+$
EIACO⇔MIACO	$-$	$s-$	$+$	$-$	$-$	$+$	$+$	$+$	$-$	$s-$	$+$	$+$
$f = 50$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	$+$	$-$	$-$	$-$	$s-$	$s-$	$s-$	$s-$	$-$	$s-$	$s-$	$s-$
RIACO⇔MIACO	$+$	$-$	$-$	$s-$	$s-$	$s-$	$s-$	$s-$	$-$	$s-$	$s-$	$s-$
EIACO⇔MIACO	$-$	$+$	$+$	$-$	$-$	$-$	$+$	$-$	$+$	$+$	$+$	$+$
$f = 100$, $m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
RIACO⇔EIACO	$+$	$-$	$+$	$s-$	$s-$	$s-$	$s-$	$s-$	$-$	$s-$	$s-$	$s-$
RIACO⇔MIACO	$+$	$-$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
EIACO⇔MIACO	$-$	$+$	$+$	$-$	$-$	$+$	$+$	$+$	$-$	$-$	$-$	$+$

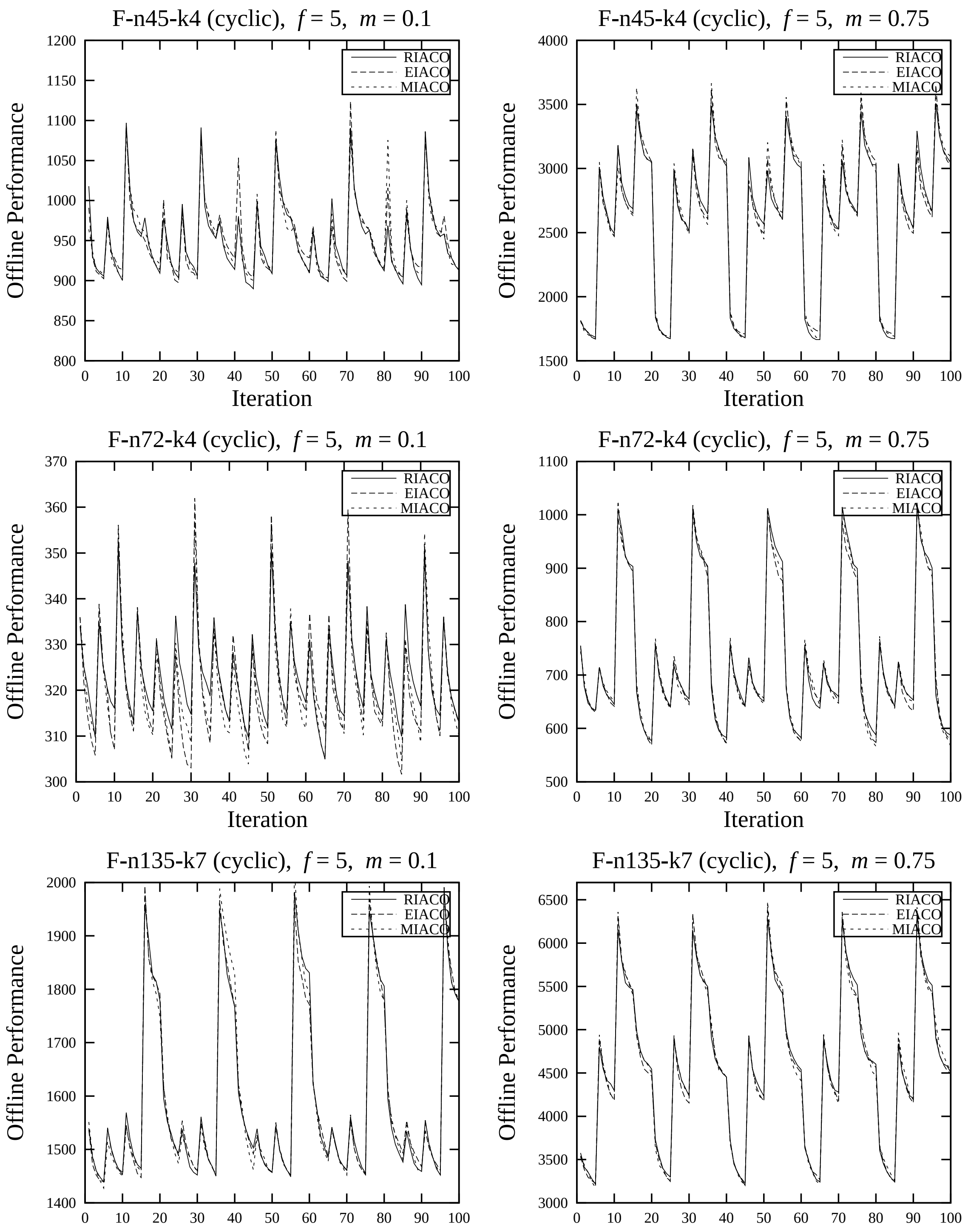


Figure 8.14: Dynamic behaviour of the algorithms for cyclic DVRPs with fast changing environments.

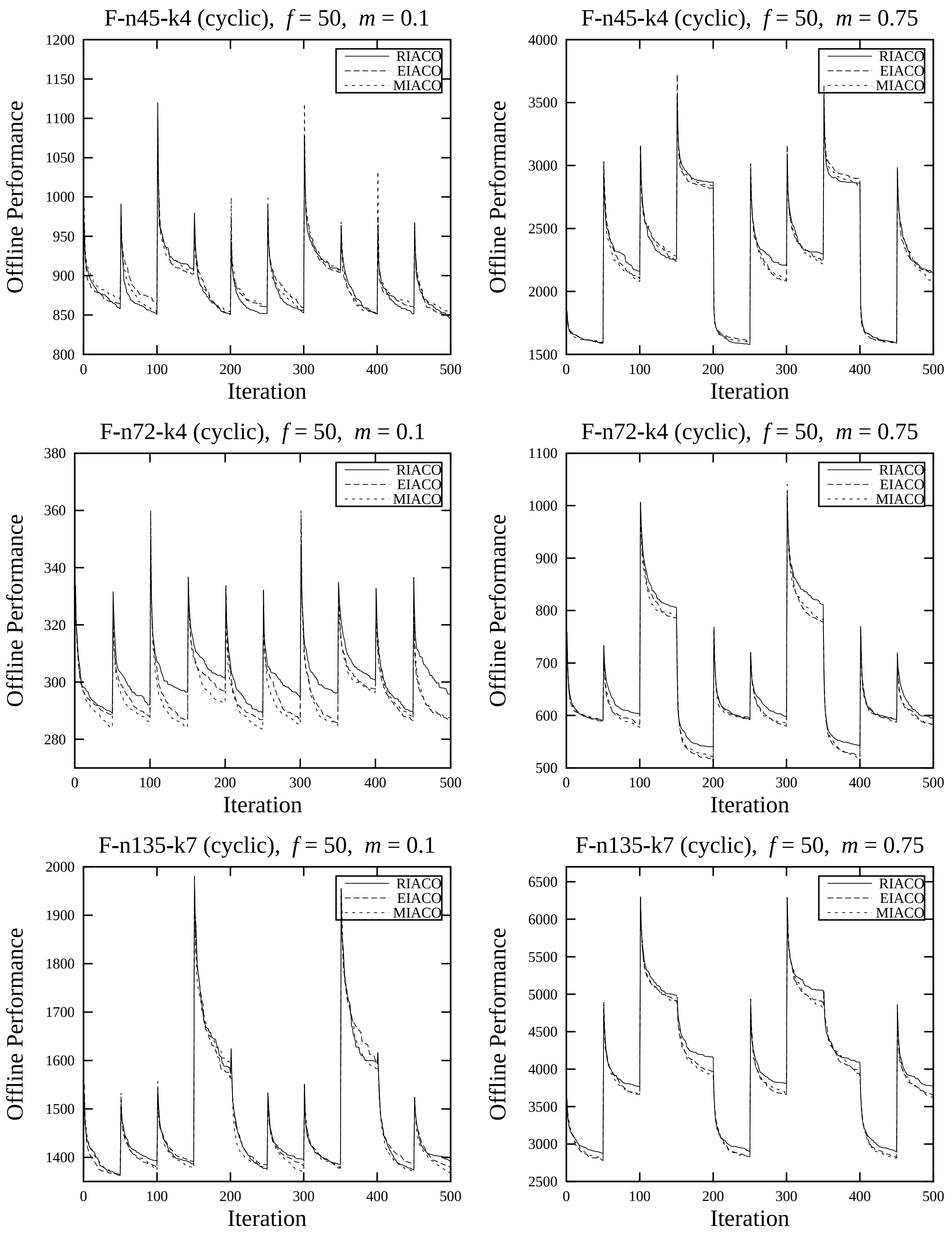


FIGURE 8.15: Dynamic behaviour of the algorithms for cyclic DVRPs with slowly changing environments.

as in the DTSP, because of the capacity constraint in DVRPs that changes the solution representation.

8.3.3.5 Effect of Immigrants Replacement Rate r

In order to investigate the effect of the immigrants replacement rate, further experiments were performed on the same problem instances with the same parameters used before but with different immigrant replacement rates, i.e., $r \in \{0.0, 0.1, 0.3, 0.5, 0.8\}$, where $r = 0.0$ means that no immigrants are generated to replace ants in the $k_{short}(t)$. RIACO and EIACO (or MIACO) are applied on the dynamic cases that showed good performance from the basic experiments above, which make use of random and guided immigrants, respectively.

In Table 8.17 and Figure 8.16, RIACO is applied on fast changing environments which shows that random immigrants improve the performance of ACO slightly. This shows that random immigrants do not have a significant effect on fast and significantly changing environments, except in F-n45-k4, different from what was observed in the DTSP. In most dynamic cases, random immigrants disturb the optimization process, as in the DTSP, otherwise the improvement is slight. For example, the improvement from RIACO($r = 0.0$) to RIACO($r = 0.3$) in F-n45-k4 is often greater than the improvement from RIACO($r = 0.0$) to RIACO($r = 0.3$) in F-n135-k7 for $m = 0.75$, whereas in F-n72-k4 the performance is degraded. In fact, this is the case for the remaining dynamic test cases, e.g., when $m = 0.1$, $m = 0.25$ and $m = 0.5$. This probably confirms our observation in the basic experiments that random immigrants have a higher probability to hit the optimum in a smaller search space and the risk of randomization decreases.

In Table 8.18 and Figure 8.16, EIACO is applied on slowly changing environments which shows that guided immigrants improve the performance of ACO in all cases.

Table 8.17: Offline performance of the proposed framework with different replacement rates of random immigrants on fast changing DVRPs. Bold values indicate the best results.

$f = 5, m \Rightarrow$	0.1	0.25	0.5	0.75
Alg. & Inst.		F-n45-k4		
RIACO($r = 0.0$)	915.7	1139.1	1867.9	3188.1
RIACO($r = 0.1$)	915.6	1139.6	**1846.7**	3186.4
RIACO($r = 0.3$)	913.7	1137.0	1859.3	**3176.4**
RIACO($r = 0.5$)	911.4	**1136.7**	1858.4	3178.9
RIACO($r = 0.8$)	**910.9**	1140.2	1866.3	3181.7
Alg. & Inst.		F-n72-k4		
RIACO($r = 0.0$)	326.2	416.3	**680.1**	**1155.8**
RIACO($r = 0.1$)	**325.9**	**415.9**	681.0	1156.2
RIACO($r = 0.3$)	326.9	417.4	683.6	1160.8
RIACO($r = 0.5$)	328.0	418.3	685.5	1166.5
RIACO($r = 0.8$)	330.4	420.9	690.4	1174.5
Alg. & Inst.		F-n135-k7		
RIACO($r = 0.0$)	1558.3	**2010.7**	3261.9	5410.3
RIACO($r = 0.1$)	**1557.9**	2011.2	**3254.7**	5409.9
RIACO($r = 0.3$)	1562.5	2015.2	3256.9	**5408.6**
RIACO($r = 0.5$)	1566.9	2019.9	3260.7	5417.9
RIACO($r = 0.8$)	1575.8	2028.0	3268.2	5434.5

In contrast with the effect of guided immigrants on DTSPs, when the replacement rate is $r > 0.3$, the performance is not degraded because of too much knowledge transferred, but is often improved on DVRPs.

The most interesting observation is when the results of EIACO($r = 0.0$), which is our proposed framework without immigrants, are compared with the corresponding results of ACS-DVRP in Table 8.12, since they were executed on the same dynamic

TABLE 8.18: Offline performance of the proposed framework with different replacement rates of elitism-based immigrants on slowly changing DVRPs. Bold values indicate the best results.

$f = 100, m \Rightarrow$	0.1	0.25	0.5	0.75
Alg. & Inst.		F-n45-k4		
EIACO($r = 0.0$)	883.1	1038.7	1624.5	2872.4
EIACO($r = 0.1$)	882.7	1039.0	1623.6	2876.4
EIACO($r = 0.3$)	875.7	1023.0	**1570.5**	2790.3
EIACO($r = 0.5$)	872.9	**1017.0**	1571.2	2772.5
EIACO($r = 0.8$)	**870.2**	1017.3	1571.1	**2747.4**
Alg. & Inst.		F-n72-k4		
EIACO($r = 0.0$)	294.0	356.1	583.9	989.0
EIACO($r = 0.1$)	297.7	356.1	584.3	988.0
EIACO($r = 0.3$)	288.7	**349.8**	572.4	959.7
EIACO($r = 0.5$)	287.8	350.4	**563.8**	959.1
EIACO($r = 0.8$)	**287.3**	351.2	565.2	**961.4**
Alg. & Inst.		F-n135-k7		
EIACO($r = 0.0$)	1436.2	1723.2	2612.9	4619.2
EIACO($r = 0.1$)	1433.3	1724.2	2612.2	4628.1
EIACO($r = 0.3$)	1418.3	1702.2	2556.2	4512.7
EIACO($r = 0.5$)	**1417.9**	**1701.7**	2552.6	4463.8
EIACO($r = 0.8$)	1420.5	1706.1	**2547.3**	**4446.1**

environments. In all dynamic cases, our proposed framework performs better than ACS-DVRP, even with the best evaporation rate. For example, the performance of EIACO($r = 0.0$) in F-n45-k4 when $m = 0.1$, $m = 0.25$, $m = 0.5$ and $m = 0.75$ is 883.1, 1038.7, 1624.5 and 2872.4, respectively, whereas the performance of ACS-DVRP with the best ρ on the same dynamic test cases is 920.2, 1134.7, 1862.9 and 3291.7, respectively.

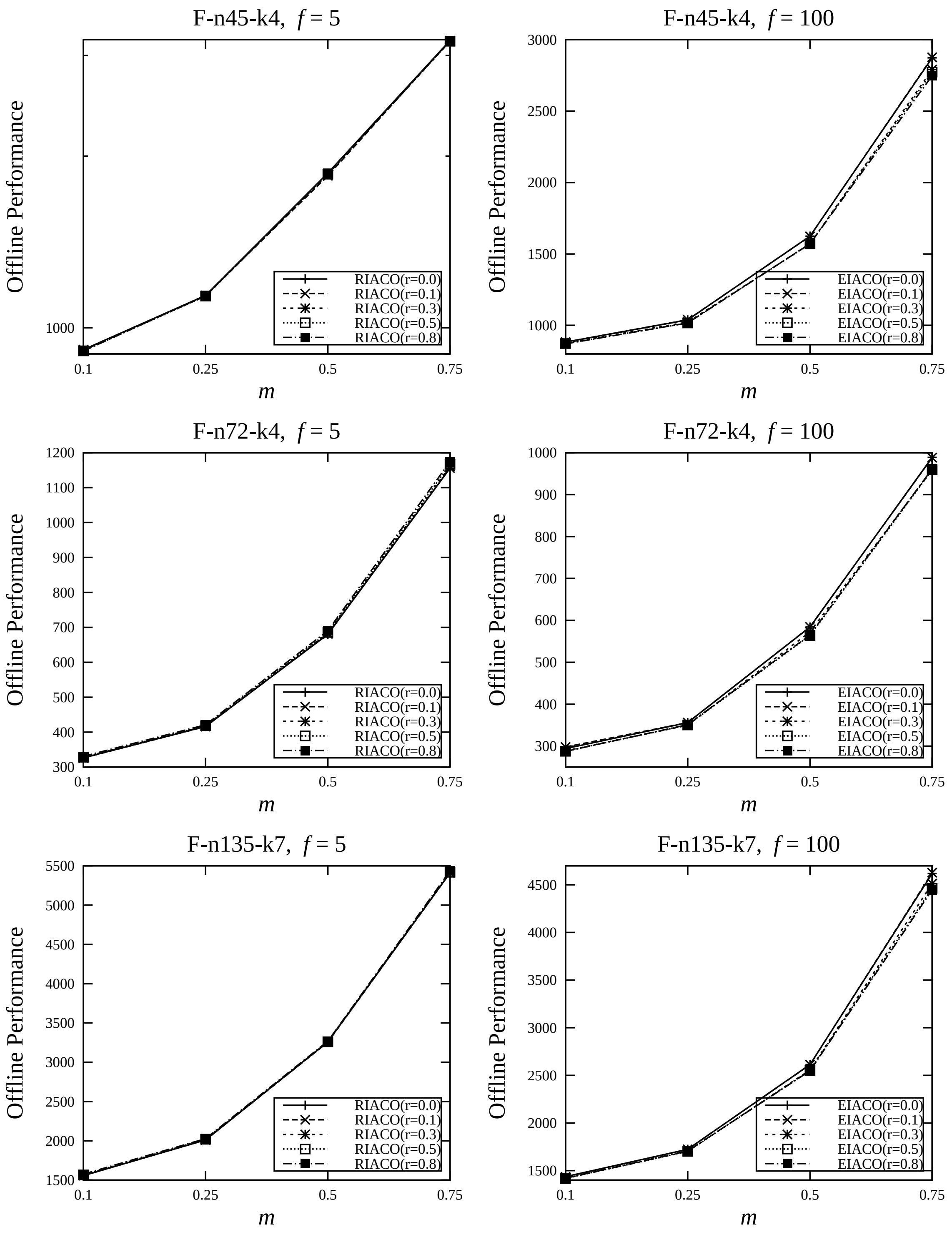


FIGURE 8.16: Offline performance of varying the replacement rate of immigrants, i.e., random immigrants (left) and guided immigrants (right), on different DVRPs.

8.3.3.6 Comparison with other Peer ACO Algorithms in DVRP

In the experiments above, the performance of RIACO, EIACO and MIACO was investigated under two kinds of dynamic environments, i.e., random and cyclic, but with fixed f and m. As in the DTSP, further experiments were performed on a DVRP with varying f and m values randomly generated with a uniform distribution in $[1, 100]$ and $[0, 1]$, respectively.

We compare the proposed algorithms with other peer ACO algorithms, i.e., ACS-DVRP, and M-ACO described in Section 6.6.2. The parameter settings for RIACO, EIACO and MIACO are the same as in the basic experiments above, whereas the rest are shown in Appendix C.

The experimental results regarding the offline performance of the aforementioned algorithms are presented in Table 8.19 with the corresponding statistical results. In Figure 8.17, the dynamic behaviour of the algorithms is presented with the corresponding m values in Figure 8.18. From the experimental results several observation can be drawn.

First, EIACO and MIACO perform similarly, and they are significantly better than other peer ACO algorithms, followed by RIACO. This matches our previous experimental results where it is better to generate guided immigrants instead of random immigrants in most dynamic cases.

Second, all the proposed algorithms perform significantly better than ACS-DVRP. This shows that immigrants schemes improve the performance of ACO in DVRPs. This is due to the pheromone policy of the proposed framework, where pheromone is generated on every iteration according to the knowledge of the previous environments. Moreover, M-ACO outperforms ACS-DVRP since the LS operators used in

Table 8.19: Experimental results regarding offline performance on the DVRP with $m = rand[0, 1]$ and $f = rand[1, 100]$ in C5. "$s+$" or "$s-$" indicates that the algorithm in the row is significantly better than the one in the column or the opposite and "$\sim$" indicates no significance.

Alg.	RIACO	EIACO	MIACO	ACS-DVRP	M-ACO
Offl. Perf.	4219.56	4170	4164.37	4378.99	4230.47
Std. Dev.	24.82	19.10	27.55	18.34	24.15
			Wilcoxon Sum-Rank Test		
RIACO		$s-$	$s-$	$s+$	$\sim$
EIACO	$s+$		$\sim$	$s+$	$s+$
MIACO	$s+$	$\sim$		$s+$	$s+$
ACS-DVRP	$s-$	$s-$	$s-$		$s-$
M-ACO	$\sim$	$s-$	$s-$	$s+$	

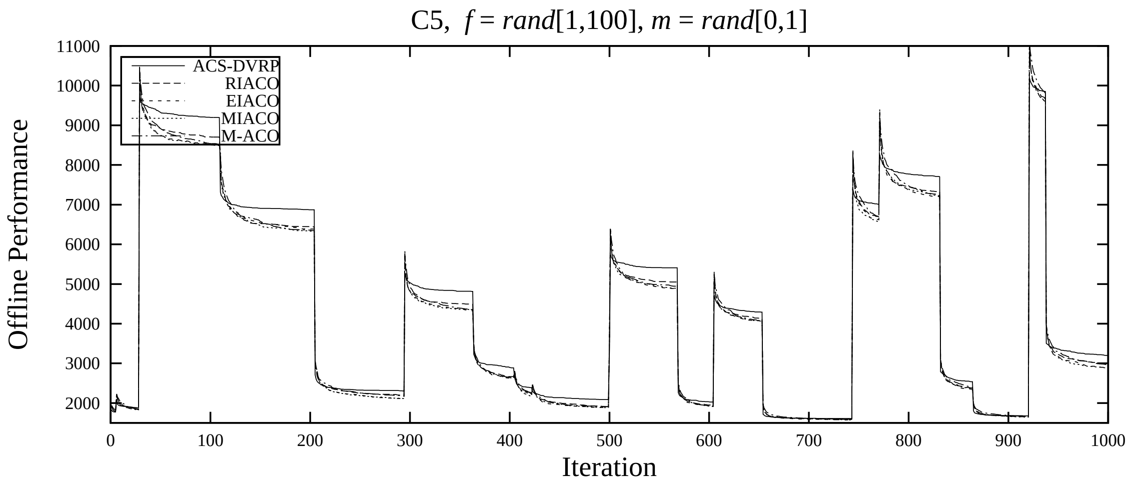


Figure 8.17: Dynamic behaviour of the proposed algorithms in comparison with other peer ACO algorithms on the DVRP with $m = rand[0, 1]$ and $f = rand[1, 100]$.

M-ACO promote exploitation and often improve the solution quality, whereas it is comparable with RIACO.

Finally, in order to investigate the effect of the diversity generated, we calculate the

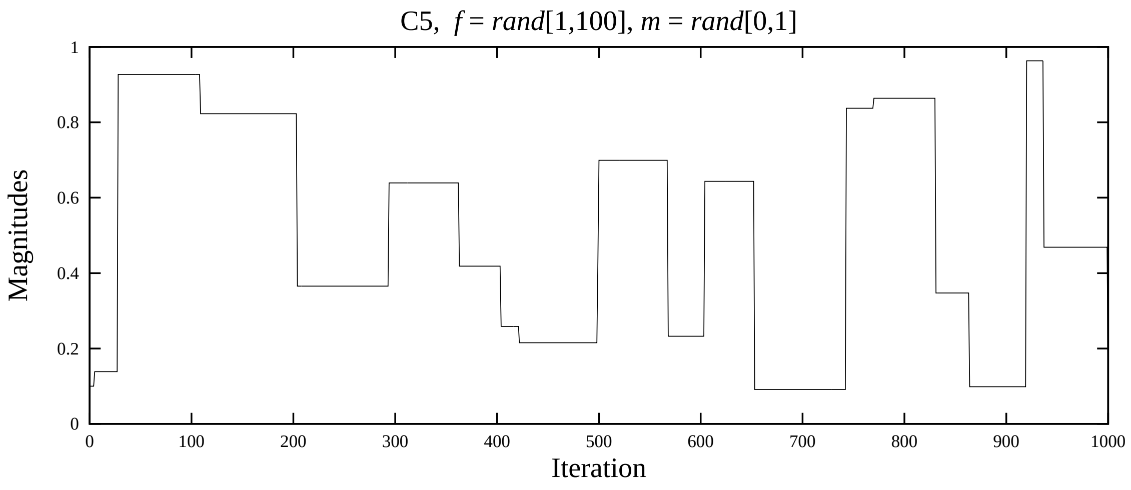


FIGURE 8.18: Varying values for $m = rand[0,1]$ and $f = rand[1,100]$ used for the DVRP.

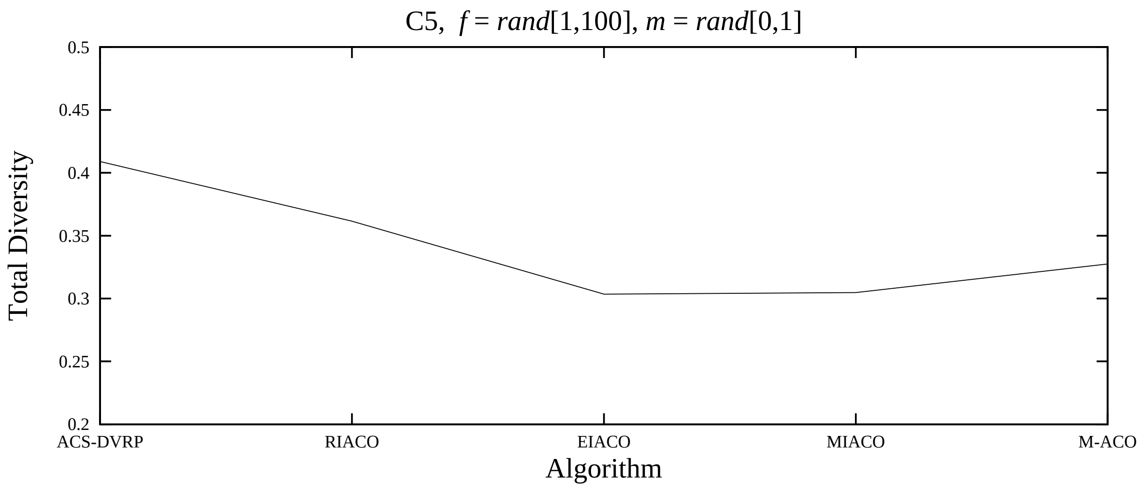


FIGURE 8.19: Total diversity of the proposed ACO algorithms in comparison with other peer ACO algorithms on the DVRP with $m = rand[0,1]$ and $f = rand[1,100]$.

total diversity using Equation 6.7. The total diversity results for the algorithms are presented in Figure 8.19. It can be observed that ACS-DVRP has a higher diversity than all other algorithms. EIACO and MIACO have the lowest diversity among all the ACO algorithms since they generate guided diversity, whereas RIACO has higher diversity from EIACO, MIACO and M-ACO. M-ACO has higher diversity than EIACO and MIACO because it uses random immigrants to balance the strong exploitation provided by the LS operators. Considering the results of the total diversity with those of the offline performance in Table 8.19 shows that ACO algorithms that maintain high diversity levels do not always achieve better performance than other ACO algorithms for the DVRP. This matches the observation found for the corresponding case of the DTSP in Section 8.3.2.8.

8.3.3.7 Summary for the DVRP Experimental Results

In this subsection we summarize all the results from the experiments of DVRP. In Figure 8.20 the results from Table 8.14 and Table 8.16 are transformed to squares, that indicate the relative performance of the algorithms. The vertical and horizontal axes indicate the magnitude and frequency values, respectively. The brightness of squares indicate how well the algorithm performs on a specific dynamic test case. For example, if an algorithm is outperformed from all its competitors in all problem instances, then the square will be white; otherwise the square will be grey. The darkness of the square depends on the number of algorithms a specific algorithm outperforms, and on the consistency in different problem instances.

In Table 8.20 the analysis of the algorithms from the experiments in DTSP is summarized, where “slow” and “fast” indicate the frequency of the dynamic environment, and where “low”, “medium” and “high” indicate the magnitude of the dynamic environment.

From Figure 8.20 it can be observed that:

1. RIACO performs well on a wide range of dynamic test cases in which there squares are lighter. This is because RIACO has an inconsistent behaviour in both random and cyclic DVRPs, e.g., performs well only on small problem instances.

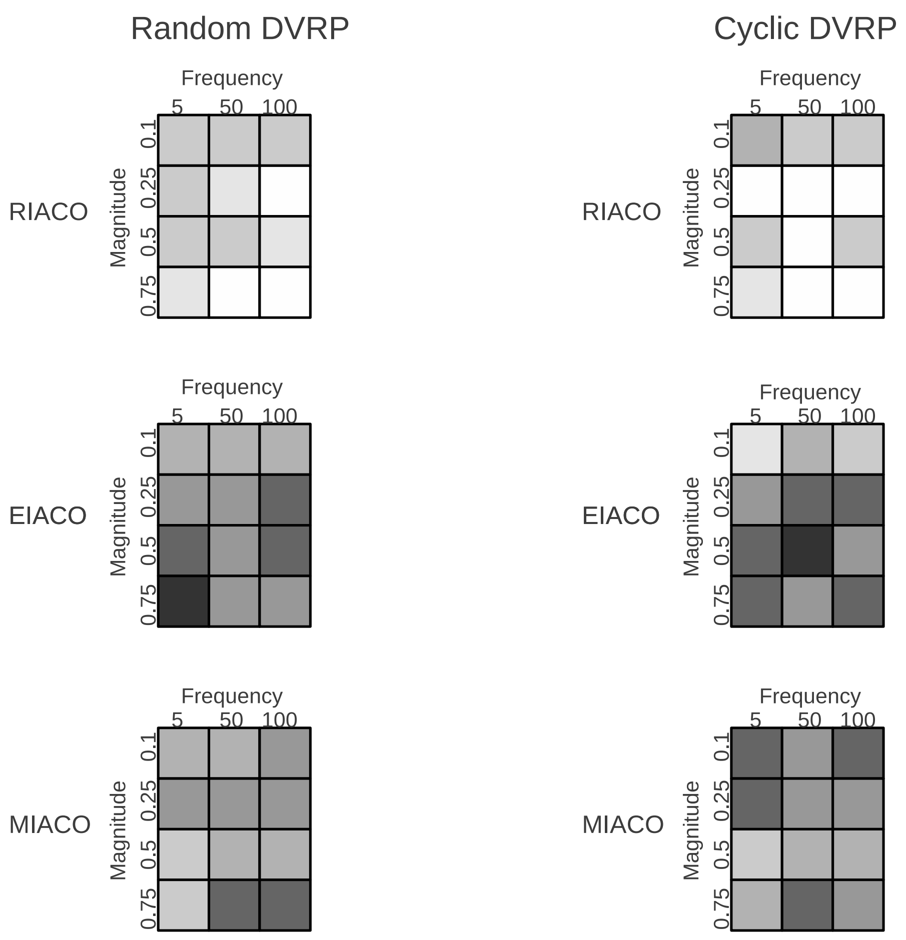


FIGURE 8.20: Performance of the algorithms on different dynamic test cases for the DVRP. The darkest squares indicate better performance of the algorithm when compared with its competitors considering all the problem instances.

2. EIACO performs well on most DVRP test cases and it is more consistent than RIACO because the squares are darker. The reason why the squares for both EIACO and MIACO are not as dark as in the DTSP cases (see Figure 8.11) is due to the inconsistency of RIACO which performs better than them in all dynamic test cases of one problem instance.

Table 8.20: Summary observations regarding the relative performance of the algorithms in different dynamic test cases of DVRPs.

	Environmental Dynamic Case				
Algorithms	Slow	Fast	Low	Medium	High
RIACO	Disturbs the optimization process except in small problem instances.	Performs better only in small problem instances which shows inconsistent behaviour.	Similar performance as in the "slow" case.	Similar performance as in the "slow" case.	Similar performance as in the "slow" case.
EIACO	Performs well since it has enough time to gain knowledge from previous environments except in small problem instances.	Similar performance as in the "slow" case.	Performs better only in larger problem instances.	Similar performance as in the "low" case.	Similar performance as in the "low" case.
MIACO	Similar performance with EIACO in both random and cyclic environments.	Similar performance as in the "slow" case.	Similar performance as in the "slow" case.	Similar performance as in the "slow" case.	Similar performance as in the "slow" case.

3. MIACO performs well in most DVRP test cases and it has a similar performance with EIACO. Moreover, the performance of MIACO has no significance difference from random DVRPs to cyclic DVRPs as in the DTSP case; see Figure 8.11. A similar observation can be made for the EIACO performance.

8.4 Summary

In this chapter we have presented our proposed algorithms that integrate immigrants schemes with an ACO framework. In every iteration, immigrant ants are generated either random or guided to replace a small portion of the worst in the current ant population. The proposed ACO framework consists of a k_{short} of size K_s, where in every iteration the K_s best ants replace all the ants of the previous iteration.

The proposed algorithms showed good performance in both DTSP and DVRP in comparison with other peer ACO algorithms. Each proposed algorithm showed good performance on different dynamic cases. In the DTSP, RIACO performed better in fast and significantly changing environments, EIACO (and MIACO) in slowly and slightly changing environments and MIACO in environments that re-appear. In the DVRP, the results where quite different. RIACO performed better in small problem instances, EIACO (and MIACO) in most dynamic cases, either random or cyclic. Moreover, the framework of the proposed algorithms performs better than conventional ACO algorithm in both DTSP and DVRP, even if no immigrants are generated.

In general, it is important to achieve a good balance between exploration and exploitation, such as the example of M-ACO, and also to achieve a good balance between the diversity generated and the knowledge transferred, such as the example of EIACO and MIACO. For the case of traditional ACO algorithms, it is important to choose a higher evaporation rate that will eliminate the pheromone trails of the previous environment quickly, in order for the ants to adapt to the new environment.

The benchmark generator with unknown optimum from Chapter 7 has been used in this chapter. Further experiments with the proposed DBGP with known optimum is a good direction for future work (for more information see Chapter 9).

Chapter 9

Conclusions and Future Work

In this thesis, we have studied the behaviour of ACO algorithms for static and dynamic COPs. These algorithms are inspired from the foraging behaviour of real ant colonies in nature. They are classified into the category of metaheuristics which are able to provide the global optimum (or a near global optimum) solution for $\mathcal{NP}$-complete problems efficiently. Often, metaheuristics so far are analyzed empirically due to the limited foundation of theoretical work.

Usually, ACO algorithms suffer from stagnation behaviour, where ants construct the same solution from early stages of the algorithm. This degrades the solution quality provided by ACO algorithms in static and dynamic environments because they may get trapped in local optima and may lose their adaptation capabilities, respectively.

9.1 Technical Contributions

Our contributions in the field of ACO cover the development of strategies that: (1) help to escape local optima in static environments; (2) enhance the adaptation capabilities in dynamic environments; and (3) the development of tools for the empirical analysis of ACO (or other metaheuristics) in dynamic permutation-encoded COPs. The main contributions are summarized as follows:

- The development of a novel scheme based on direct communication for ACO, which delays stagnation behaviour and avoids possible local optima in the TSP. The results showed that ACO algorithms with the DC scheme produce better performance than conventional ACO algorithms.
- The DC scheme has been extended and applied to the VRP with similar good performance as in the TSP.
- The development of a novel ACO framework to address dynamic environments. The framework prepares the pheromone trails in every iteration and speeds up the adaptation capabilities of the algorithm. The conventional ACO framework eliminates pheromone trails with evaporation (according to the rate), whereas in the proposed one, pheromone trails are eliminated directly. From the results, the proposed framework performs better than the conventional ACO for DTSPs.
- The novel integration of immigrants schemes with the proposed ACO framework. RIACO, EIACO, and MIACO showed good performance in different dynamic cases for DTSPs and DVRPs.
- RIACO, EIACO and MIACO have been extended and applied to the DVRP with similar performance.
- The development of a benchmark generator that generates dynamic environments with different properties from static problem instances. It works for both DTSPs and DVRPs, and adds traffic factors to the problem instances in order to model a real-world application.

- The development of another general benchmark generator for COPs that are represented with weighted graphs. DBGP generates dynamic environments, where the optimum remains unchanged (if known), because it moves the population to a different position in the fitness landscape instead of modifying it.

9.2 Conclusions of the Experimental Results

From the experimental results presented in Chapters 5 and 8, for the ACO with DC scheme in static environments and ACO with immigrants schemes in dynamic environments, respectively, several concluding remarks can be drawn as follows:

- The DC scheme helps the population to escape possible local optima and improves the performance of conventional ACO in static TSPs and VRPs, with an appropriate communication range.

- Increasing the evaporation rate improves the performance of $\mathcal{MM}$AS in DTSPs, whereas increasing the evaporation rate improves the performance of ACS-DVRP only in large problem instances in DVRPs.

- As the problem size and the magnitude of change increases the performance of conventional ACO algorithms is degraded, when compared with the proposed framework, in which a short-term memory is used, in dynamic environments.

- Each proposed algorithm performs good on different DTSP test cases. RIACO performs better in fast and significantly changing environments. EIACO (and MIACO) performs better in slowly and slightly changing environments. MIACO performs better in environments that re-appear.

- Each proposed algorithm performs good on different DVRP test cases. RIACO performs better in small problems. EIACO (and MIACO) performs better in most dynamic cases, even if the environments re-appear or not.
- The performance of RIACO, EIACO and MIACO is not consistent on different dynamic COPs because they have different performance in some dynamic test cases of DTSPs when compared with the corresponding test cases of DVRPs. This is probably because the proposed algorithms were initially designed to address different DTSPs, and they were modified to address DVRPs.
- The size of the long-term memory used in MIACO is not dependent on the number of states of the cyclic dynamic environment.
- The effect of the immigrants replacement rate for the proposed algorithms depends on the dynamic COP and the magnitude of change.
- Maintaining high diversity levels does not always achieve better performance than other ACO algorithms in both DTSPs and DVRPs.
- It is important to achieve a good balance between exploration and exploitation, and also to achieve a good balance between the diversity generated and the knowledge transferred in DOPs.

Generally speaking, ACO with DC scheme outperform other peer ACO algorithms in TSPs and VRPs and ACO with immigrants schemes outperform other peer ACO algorithms in almost all dynamic test cases in DTSPs and DVRPs.

9.3 Future Work

The main goals of the thesis was the improvement of ACO in static and dynamic environments. From our experiments and observations, several future works and directions are posed, many of which were pointed out across the thesis, and they are briefly summarized in the next subsections.

9.3.1 ACO in Static Environments

The future work for ACO in static environments is summarized as follows:

- It will be interesting to apply the DC scheme with other ACO algorithm variations and with other variations of TSPs and VRPs.
- Investigate the effect of the DC scheme when an ACO algorithm is applied with an LS, e.g., the 2-opt operator [164]. Usually, on larger problem instances, the solution quality of algorithms is more likely to be degraded, whereas a local search may improve it at the price of more computation time. Therefore, DC may be able to guide the local search operator for better solution quality and computation time.
- Integrate crossover and mutation operators to the DC scheme, for the ants to exchange information, instead of the swap operator.

9.3.2 ACO in Dynamic Environments

Different from ACO in static environments, there are more directions for future work for ACO in dynamic environments, and they are summarized as follows:

- Most of the strategies used to develop RIACO, EIACO and MIACO are inspired from EAs for DOPs. However, they have been developed for DOPs with a binary search space. For example, the memory scheme [223] can be further improved in order to address more accurately DOPs with combinatorial space.

- The effect of immigrants schemes usually depends on the magnitude of change in the environment and the optimization period of the algorithm. In order to achieve a good balance between exploration and exploitation it will be interesting to adapt the replacement rate of the immigrants schemes [289].

- The performance of conventional ACO can be further improved in DOPs if the evaporation rate, i.e., ρ, is adapted, since it has a significant impact on ACO's performance. Different evaporation rates, either high or low, may be the best choice in different periods of the algorithm's execution.

- Apart from single-colony ACO, multi-colony ACO can also be applied to DOPs since different colonies may explore different areas in the search space. RI-ACO, EIACO and MIACO can be extended as multi-colony algorithms and can exchange immigrants between the colonies. In fact, immigrants have been initially proposed for multi-population algorithms [269, 270].

- More real-world related models can be modelled with the proposed benchmark generators for DOPs. For example, to integrate the time-linkage property where the future behaviour of the problem depends on the current or previous solution found by the algorithm [201].

- Moreover, it will be interesting to integrate the DBGP benchmark generator with the ROOT framework [90, 287]. In fact, some real-world dynamic (combinatorial) problems, in scheduling and vehicle routing, have the time-linkage property [19]. Therefore, tracking the moving optimum, as in our experiments, may not be the best choice, because the decision made to improve the performance at present may influence the performance in the future. Hence, the overall performance may be degraded in the long run.

9.3.3 Further Discussion Regarding the Relation of Dynamic Optimization and Real-World Applications

From the surveys regarding the ACO applications in static and dynamic environments in Tables 4.1 and 6.1, respectively, it can be observed that the work in the former one is extensive, whereas in the latter one the work is still in its infancy. Moreover, the amount work on ACO for DOPs is also relatively weak in comparison with the work done on EAs for DOPs. Since ACO algorithms showed that they have good performance in DTSPs and DVRPs it will be interesting to apply them on more realistic applications, or even make a start on theoretical works regarding ACO for simple DOPs, as done for EAs [225]. Of course, this thesis has focused on the practical side of the ACO and not on the theoretical side.

Although the research of EDO (or more generally dynamic optimization) is mature enough it still has many aspects for further consideration, which are summarized as follows:

- Define a computational experimentation protocol to compare the performance of algorithms in dynamic environments.
- Develop new performance measurements to tackle the performance of algorithms more accurately in dynamic environments.
- Consider other types of dynamism to perfectly fit real-world application.
- Consider multi-objective dynamic optimization, since many real-world problems are not only dynamic, but they may have several objectives [138, 166].
- Develop methods to predict dynamic changes or the severity of the dynamic change, e.g., integrate learning methods with metaheuristics.

In nature-inspired metaheuristics, there is a great gap between theory and practice in academic research, since many applications have been proposed and there are just a few theoretical works. With the term "practice" some researchers mean that the metaheuristic actually works and has been tested in a real-world scenario, whereas other researchers mean that the metaheuristic has been applied on academic benchmarks that model some characteristics of a real-world scenario. To the best of our knowledge, the practical works, with the former meaning, are just a few, which is a similar case with theoretical works, whereas there are many practical works, with the latter meaning. Therefore, the statement regarding the great gap between theory and practice differs depending on the meaning of the term practice.

In case metaheuristics are applied on academic benchmarks, some evidence can be obtained that they might work on the actual real-world scenarios that the academic benchmarks model, but there is no guarantee that they will have the same good performance. Therefore, it is essential to apply effective metaheuristics in actual real-world scenarios to bring the academic research of dynamic optimization closer to real-world applications.

Appendix A

Description of Benchmarks

Static TSP instances

Instance Name	n	Known Optimum
eil51	51	426
eil76	76	538
eil101	101	629
kroA100	100	21282
kroA150	150	26524
kroA200	200	29368
lin318	318	42029
pcb442	442	50778
att532	532	27686
rat783	783	8806
pcb1173	1173	56892
pr2392	2392	378032

Dynamic TSP instances

Instance Name	n	Known Optimum
pr76	76	108159
pr152	152	73682
pr299	299	48191

Static VRP instances

Instance Name	n	Q	L	δ	v	Best Known Optimum
Random Problems						
C1	50	160	∞	0	5	524.61
C2	75	140	∞	0	10	835.26
C3	100	200	∞	0	8	826.14
C4	150	200	∞	0	12	1028.42
C5	199	200	∞	0	16	1291.29
C6	50	160	200	10	6	555.43
C7	75	140	160	10	12	909.68
C8	100	200	230	10	9	865.94
C9	150	200	200	10	14	1162.55
C10	199	200	200	10	19	1395.85
Clustered Problems						
C11	120	200	∞	0	9	1042.11
C12	100	200	∞	0	10	819.56
C13	120	200	720	50	12	1541.14
C14	100	200	1040	90	11	866.37

Dynamic VRP instances

Instance Name	n	Q	v	Best Known Optimum
F-n45-k4	45	2010	4	724
F-n72-k4	72	30000	4	237
F-n135-k7	135	2210	7	1162

Appendix B

Parameter Settings for Static Combinatorial Problems

TSP parameters

ACO algorithm	α	β	ρ	μ	τ_0
AS	1	5	0.5	25	μ/C^{nn}
EAS	1	5	0.5	25	$(e+\mu)/\rho C^{nn}$
AS$_{rank}$	1	5	0.1	25	$0.5r(r-1)/\rho C^{nn}$
$\mathcal{MM}$AS	1	5	0.2	25	$1/\rho C^{nn}$
ACS	–	5	0.1	10	$1/nC^{nn}$
BWAS	1	5	0.1	25	$1/nC^{nn}$
$\mathcal{MM}$AS+DC	1	5	0.2	25	$1/\rho C^{nn}$

The additional parameters for the ACO algorithms are set as follows:

- EAS: The parameter e is set to $e = \mu$.

- AS$_{rank}$: The parameter w is set to $w = 6$.
- $\mathcal{MM}$AS: The pheromone trail boundaries are set to $\tau_{max} = 1/\rho C^{bs}$ and $\tau_{min} = (1 - \sqrt[n]{0.05})/((avg - 1) \times \sqrt[n]{0.05})$ (see [245] for more details).
- ACS: The parameter ξ is set to $\xi = 0.1$ and q_0 is set to $q_0 = 0.9$.
- BWAS: The parameters σ and p_{phmut} are set to $\sigma = 0.4$ and $p_{phmut} = 0.3$, respectively. Moreover, q_0 is set to $q_0 = 0.9$.
- $\mathcal{MM}$AS+DC: The parameters ω and T_r are set to $\omega = 0.5$ and T_r ranges $0.6 \leq T_r \leq 0.8$ depending on the size of the problem instance, respectively.

VRP parameters

ACO algorithm	α	β	ρ	μ	τ_0	q_0
AS$_{rank}$-CVRP	1	5	0.1	50	$0.5r(r-1)/\rho C^{nn}$	0.0
MACS-VRP	–	5	0.1	50	$1/nC^{nn}$	0.9
AS$_{rank}$-CVRP+DC	1	5	0.1	50	$0.5r(r-1)/\rho C^{nn}$	0.0

- AS$_{rank}$-CVRP: The parameter w is set to $w = 6$.
- AS$_{rank}$-CVRP+DC: The parameters ω and T_r are set to $\omega = 0.5$ and $T_r = 0.8$, respectively.

Appendix C

Parameter Settings for Dynamic Combinatorial Problems

TSP parameters

ACO algorithm	α	β	q_0	ρ	K_s	K_l	μ
RIACO	1	5	0.0	–	6	–	28
EIACO	1	5	0.0	–	6	–	28
MIACO	1	5	0.0	–	6	3	25
$\mathcal{MM}$AS	1	5	0.0	0.6	–	–	28
P-ACO	1	5	0.9	–	–	3	28
M-ACO	1	5	0.9	–	–	3	20
SC-PACO	1	5	0.5	–	–	8	20
FS-PACO	1	5	0.5	–	–	8	28

The additional parameters for some algorithms are set as follows:

- RIACO, EIACO and MIACO: The parameter r is set to $r = 0.3$.

- M-ACO: The parameter LS_{steps} is set to $LS_{steps} = 8$.
- FS-PACO: The parameters σ_{share}, a, and λ are set to $\sigma_{share} = 0.2$, $a = 1$ and $\lambda = 2$, respectively.
- SC-PACO: The parameter λ is set to $\lambda = 2$.
- $\mathcal{MM}$AS: The pheromone trail boundaries are set to $\tau_{max} = 1/\rho C^{bs}$ and $\tau_{min} = (1 - \sqrt[n]{0.05})/((avg - 1) \times \sqrt[n]{0.05})$ (see [245] for more details).

VRP parameters

ACO algorithm	α	β	q_0	ρ	K_s	K_l	μ
RIACO	1	5	0.0	–	6	–	28
EIACO	1	5	0.0	–	6	–	28
MIACO	1	5	0.0	–	6	3	25
ACS-VRP	–	5	0.9	0.2	–	–	28
M-ACO	1	5	0.0	–	–	3	20

The additional parameters for some algorithms are set as follows:

- RIACO, EIACO and MIACO: The parameter r is set to $r = 0.3$.
- M-ACO: The parameter LS_{steps} is set to $LS_{steps} = 8$.
- ACS-DVRP: The parameter ξ is set to $\xi = 0.1$.

Bibliography

[1] M.H. Afshar. A new transition rule for ant colony optimization algorithms: application to pipe network optimization problems, *Engineering Optimization*, 37(5), pp. 525–540, Taylor & Francis, 2005.

[2] D.A. Alexandrov, Y.A. Kochetov. The behavior of the ant colony algorithm for the set covering problem. *Proceedings of the 1999 Operations Research*, K. Inderfurth, G. Schwodiauer, W. Domschke, F. Juhnke, P. Kleinschmidt, G. Wäscher, editors, pp. 255–260. Springer-Verlag, 2000.

[3] D. Angus. Niching for population-based ant colony optimization. *Proceedings of the 2nd International IEEE Conference on e-Science and Grid Computing, Workshop on Biologically-inspired Optimisation Methods for Parallel and Distributed Architectures: Algorithms, Systems and Applications*, 2006.

[4] D. Angus, T. Hendtlass. Dynamic ant colony optimisation. *Applied Intelligence*, 23, pp. 33–38, 2005.

[5] D.L. Applegate, R.M. Bixby, V. Chvátal, W.J. Cook. *The Traveling Salesman Problem*, 2006.

[6] T. Bäck. On the behaviour of evolutionary algorithms in dynamic environments, *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pp. 446–451, IEEE Press, 1998.

[7] A. Bauer, B. Bullnheimer, R. F. Hartl, C. Strauss. An ant colony optimization approach for the single machine total tardiness problem. *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, pp. 1445–1450. IEEE Press, 1999.

[8] S. Baluja, R. Caruana. Removing the genetic from the standard genetic algorithm. *Proceedings of the 12th International Conference on Machine Learning*, A. Prieditis, S. Russell, editors, pp. 38–46, Morgan Kaufmann, 1995.

[9] M. Bandieramonte, A. Di Stefano, G. Morana. An ACO inspired strategy to improve jobs scheduling in a grid environment, *Proceedings of the 8th International Conference on Algorithms and Architectures for Parallel Processing*, LNCS 5022, pp. 30–41, Springer-Verlag, 2008.

[10] R. Beckers, J.-L. Deneubourg, S. Gross. Modulation of trail laying in the ant lasius niger (hymenopetra:Formicidae) and its role in the collective selection of a food source. *Journal of Insect Behaviour*, 6(6), pp. 751–759, 1993.

[11] J.E. Bell, P.R. McMullen. Ant colony optimization techniques for the vehicle routing problem, *Advanced Engineering Informatics*, 18(1), pp. 41–48, Elsevier Science, 2004.

[12] L. Bianchi, L.M. Gambardella, M.Dorigo. An ant colony optimization approach to the probabilistic traveling salesman problem, *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, LNCS 2439, pp. 883–892, Springer-Verlag, 2002.

[13] L. Bianchi, L.M. Gambardella, M.Dorigo. Solving the homogeneous probabilistic travelling salesman problem by the ACO metaheuristic. *Proceedings of the Third International Workshop on Ant Algorithms*, M. Dorigo, G. Di Caro, M. Sampels, editors, LNCS 2463, pp. 176–187, Springer-Verlag, 2002.

[14] C. Blum. ACO applied to group shop scheduling: a case study on intensification and diversification. *Proceedings of the Third International Workshop on Ant Algorithms*, M. Dorigo, G. Di Caro, M. Sampels, editors, LNCS 2463, pp. 14–27. Springer-Verlag, 2002.

[15] C. Blum. Beam-ACO - hybridizing ant colony optimization with beam search: an application to open shop scheduling, *Computers and Operations Research*, 32(6), pp. 1565–1591, Elsevier Science, 2005.

[16] C. Blum, K. Socha. Training feed-forward neural networks with ant colony optimization: An application to pattern classification, *Proceedings of the 5th International Conference on Hybrid Intelligent Systems*, pp. 233-238, 2005.

[17] E. Bonabeau, M. Dorigo, G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, New York, 1999.

[18] E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, G. Theraulaz. Routing in telecommunication networks with Smart ant-like agents. *Proceedings of the Second International Workshop on Intelligent Agents for Telecommunication Applications*, LNAI 1437, Springer-Verlag, 1998.

[19] P.A.N. Bosman. Learning, anticipation and time-deception in evolutionary online dynamic optimization, *Proceedings of the 2005 International Conference on Genetic and Evolutionary Computation*, pp. 39–47, ACM Press, 2005.

[20] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. *Proceedings of the 199 IEEE Congress on Evolutionary Computation*, 3, pp. 1875–1882, IEEE Press, 1999.

[21] J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer, 2001.

[22] J. Branke, Schmeck H. Designing evolutionary algorithms for dynamic optimization problems, *Theory and Application of Evolutionary Computation: Recent Trends*, S. Tsutsui, A. Ghosh, editors, pp. 239–262, Springer-Verlag, 2003.

[23] J. Branke, T. Kauβler, C. Schmidt, H. Schmeck. A multi-population approach to dynamic optimization problems. *Adaptive Computing in Design and Manufacturing*, pp. 299–308, Springer-Verlag, 2000.

[24] J. Branke, E. Salihoğlu, C. Uyar. Towards an analysis of dynamic environments. *Proceedings of the 2005 International Conference on Genetic and Evolutionary Computation*, pp. 1433–1440, ACM Press, 2005.

[25] J. Branke, W. Wang. Theoretical analysis of simple evolution strategies in quickly changing environments, *Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation*, LNCS 2723, pp. 537–548, Springer-Verlag, 2003.

[26] J. Branke, M. Orbayi, S. Uyar. The role of representations in dynamic knapsack problems. *Proceedings of the 2006 EvoWorkshops on Applications of Evolutionary Computation*, LNCS 3907, pp. 764–775, Springer-Verlag, 2006.

[27] B. Bullnheimer, R. F. Hartl, C. Strauss. Applying the ant system to the vehicle routing problem. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voβ S. Martello, I. H. Osman, C. Roucairol, editors, pp. 285–296, Kluwer Academic, 1999.

[28] B. Bullnheimer, R. F. Hartl, C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89, pp. 319–328, 1999.

[29] B. Bullnheimer, R.F. Hartl, C. Strauss. A new rank-based version of the ant system: a computational study. *Central European Journal of Operations Research and Economics*, 7(1), pp. 25–38, 1999.

[30] B. Bullnheimer, G. Kotsis, C. Strauss. Parallelization strategies for the ant system, *High Performance Algorithms and Software in Nonlinear Optimization, Applied Optimization*, 24, pp. 87–100, Kluwer Academic, 1998.

[31] D. Câmara, A.F. Loureiro. A novel routing algorithm for ad hoc networks. *Proceedings of the 33rd Hawaii International Conference on System Sciences*, 2, pp. 1–8, IEEE Press, 2000.

[32] D. Câmara, A.F. Loureiro. Gps/ant-like routing in ad hoc networks. *Telecommunication Systems*,18(1–3), pp. 85–100, 2001.

[33] S. Camazine, J. Sneyd J. A model of collective nectar source by honey bees: Self-organization through simple rules. *Journal of Theoretical Biology*, 149, pp. 547–571, 1991.

[34] J. Campbell, A. Langevin. Roadway snow and ice control. *Arc routing: Theory, Solutions and Applications*, pp. 389–418, Kluwer Academic, 2000.

[35] L. Carrillo, J.L. Marzo, L. Fábrega, , P. Vilá, C. Guadall. Ant colony behaviour as routing mechanism to provide quality of service. *Proceedings of the Fourth International Workshop on Ant Algorithms*, M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, T. Stützle, editors, LNCS 3172, pp. 418–419, Springer-Verlag, 2004.

[36] W. Cedeno, V. R. Vemuri. On the use of niching for dynamic landscapes, *Proceedings of the 1997 IEEE International Conference in Evolutionary Computation*, pp. 361–366, IEEE Press, 1997.

[37] S.C. Chu, J.F. Roddick, J.S. Pan. Ant colony system with communication strategies, *Information Sciences*, 167(1–4), pp. 63–76, 2004.

[38] N. Christofides. Vehicle Routing, *The Traveling Salesman Problem.* pp. 431–448, John Wiley, 1985.

[39] S.E. Christodoulou, G. Ellinas. Pipe routing through ant colony optimization, *Journal of Infrastructure Systems*, 16(2), pp. 149–159, 2010.

[40] G. Clarke, J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4), pp. 568–581, 1964.

[41] H. G. Cobb. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments, *Technical Report AIC-90-001*, Naval Research Laboratory, Washington, USA, 1990.

[42] O. Cordón, I.F. de Viana, F. Herrera. A new ACO model integrating evolutionary computation concepts: The best-worst Ant System. *Abstract Proceedings of the 2nd International Workshop on Ant Algorithms*, pp. 22–29, 2000.

[43] O. Cordón, I.F. de Viana, F. Herrera. Analysis of the best-worst Ant System and its variants on the TSP. *Mathware and Soft Computing*, 9(2–3), pp. 177–192, 2002.

[44] O. Cordón, I.F. de Viana, F. Herrera. Analysis of the best-worst Ant System and its a variants on the qap. *Proceedings of the 3rd International Workshop on Ant Algorithms*, M. Dorigo, G. Di Caro, M. Sampels, editors, LNCS 2463, pp. 228–234, Springer-Verlag, 2002.

[45] D. Corne, M. Dorigo, F. Glover. *New Ideas in Optimization*, London, McGraw Hill, 1999.

[46] A. Colorni, M. Dorigo, V. Maniezzo. Distributed optimization by ant colonies. *Proceedings of the 1st European Conference on Artificial Life*. F.J Verela, P. Bourgine, editors, pp. 134–142, MIT Press, 1992.

[47] A. Colorni, M. Dorigo, V. Maniezzo, M. Trubian. Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science*, 34, pp. 39–53, 1994.

[48] D. Costa, A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48, pp. 295–305, 1997.

[49] C. Cruz, J. R. Gonzalez, D. A. Pelta, Optimization in dynamic environments: A survey on problems, methods and measures, *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, 15(7), pp. 1427–1448, Springer-Verlag, 2011.

[50] G.B. Dantzig, J.H Ramser. The truck dispatching problem, *Management Science*, 6(1), pp. 80–91, 1959.

[51] J. de Jong, M.A. Wiering. Multiple ant colony systems for the busstop allocation problem. *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence*, pp. 141–148, 2001.

[52] M. den Besten, T. Stützle, M. Dorigo. Ant colony optimization for the total weighted tardiness problem. *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, H. Schwefel, editors, LNCS 1917, pp. 611–620, Springer-Verlag, 2000.

[53] J.-L. Deneubourg, S. Aron, S. Goss, J.-M. Pasteels. The self-organizing exploratory pattern of the Argentine ant. *Journal of Insect Behavior*, 3, pp. 159–168, 1990.

[54] J.-L. Denebourg, J.M. Pasteels, J.C. Verhaeghe. Probabilistic behaviour in ants : a strategy of errors? *Journal of Theoretical Biology*, 105, pp. 259–271, 1983.

[55] B. De Rosa, G. Improta, G. Ghiani, R. Musmanno. The arc routing and scheduling problem with transshipment. *Transportation Science*, 36(3), pp. 301–313, 2002

[56] N.C. Demirel, M.D. Toksari. Optimization of the quadratic assignment problem using an ant colony algorithm, *Applied Mathematics and Computation*, 183(1), pp. 427–435, 2006.

[57] K.F. Doerner, M. Gronalt, R.F Hartl, M. Reimann, C. Strauss, M. Stummer. Savings ants for the vehicle routing problem. *Proceedings of 2002 EvoWorkshops on Applications of Evolutionary Computation*, LNCS 2279, pp. 11–20, Springer-Verlag, 2002.

[58] K.F. Doerner, R.F. Hartl, G. Kiechle, M. Lucka, M. Reimann Parallel ant systems for the capacitated vehicle routing problem. *Proceedings of the 4th European Conference on Evolutionary Computation in Combinatorial Optimization*, LNCS 3004, pp. 72–83, Springer-Verlag, 2004.

[59] D.D. Dong, H.Q. Dinh, H.X. Huan. On the pheromone update rules of ant colony optimization approaches for the job shop scheduling problem, *Proceedings of the 11th Pacific Rim International Conference on Multi-Agents*, 5357, pp. 153–160, 2008.

[60] G.F. Dong, W.W. Guo. A cooperative ant colony system and genetic algorithm for TSPs, *Proceedings of the 1st International Conference on Swarm Intelligence*, LNCS 6145, pp. 597–604, Springer-Verlag, 2010.

[61] F. Ducatelle, G. Di Caro, A. Förster, L.M. Gambardella. Mobile stigmergic markers for navigation in a heterogeneous robotic swarm. *Proceedings of the 7th International Conference on Swarm Intelligence*, 2010.

[62] F. Ducatelle, G. Di Caro, A. Förster, L.M. Gambardella. Cooperative self-organization in a heterogeneous swarm robotic system. *Proceedings of 2010 International Conference on Genetic and Evolutionary Computation*, pp. 87–94, ACM Press, 2010.

[63] A. V. Donati, R. Montemanni, N. Casagrande, A. E. Rizzoli, L. M. Gambardella. Time dependent vehicle routing problem with a multi ant colony system. *European Journal of Operational Research*, 185(3), pp. 1174–1191, 2008.

[64] M. Dorigo, G. Di Caro, L. M. Gambardella. Ant algorithms for discrete optimization, *Artificial Life*, 5(2), pp. 137–172, 1999.

[65] M. Dorigo, L.M. Gambardella. Ant colonies for the travelling salesman problem. *BioSystems*, 43(2), pp. 73–81, 1997.

[66] M. Dorigo, L.M. Gambardella. Ant colony system: a cooperative learning approach to the travelling salesman problem. *IEEE Transactions in Evolutionary Computation*, 1(1), pp. 53–66, IEEE Press, 1997.

[67] M. Dorigo, V. Maniezzo, A. Colorni. Positive feedback as a strategy. *Technical report 91-016*, Dipartimento di Elettronica, Politecnico di Milano, Milan, 1991

[68] M. Dorigo, V. Maniezzo, A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(1), pp. 29–41, IEEE Press, 1996.

[69] M. Dorigo, T. Stützle. *Ant Colony Optimization.* MIT Press, London, 2004.

[70] K.A. Dowsland, J.M. Thompson. Ant colony optimization for the examination scheduling problem. *Journal of the Operational Research Society*, 56(4), pp. 426–438, Palgrave Publishers, 2005.

[71] G. Di Caro, M. Dorigo. Adaptive learning of routing tables in communication networks. *Proceedings of the Italian Workshop on Machine Learning*, 1997.

[72] G. Di Caro, M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9, pp. 317–365, 1998.

[73] G. Di Caro, F. Ducatelle, L.M. Gambardella. AntHocNet: an ant-based hybrid routing algorithm for mobile ad hoc networks. *Proceedings of 8th International Conference on Parallel Problem Solving from Nature*, LNCS 3242, pp. 461–470, Springer-Verlag, 2004.

[74] G. Di Caro, F. Ducatelle, L.M. Gambardella. AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Transactions on Telecommunications*, 16(5), pp. 443–455, 2005.

[75] G. Di Caro, T. Vasilakos. Ant-SELA: Ant-agents and stochastic automata learn adaptive routing tables for QoS routing in ATM networks. *Proceedings of the 2nd International Workshop on Ant Colony Optimization*, 2000.

[76] E.W. Dijkstra. A note on two problems in connexion with graphs, *Numerische Mathematik*, 1, pp. 269–271, 1959.

[77] M. Eley. Some experiments with ant colony algorithms for the exam timetabling problem, *Proceedings of the 5th International Workshop on Ant Colony Optimization and Swarm Intelligence*, M.Dorigo, L.M. Gambardella, M. Birattari, A. Martinoli, T. Stützle, editors, LNCS 4150, pp. 492–499, Springer-Verlag, 2006.

[78] M. Eley. Ant algorithms for the exam timetabling problem, *Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling*, LNCS 3867, pp. 364–382, Springer-Verlag, 2007.

[79] L. Ellabib, P. Calamai, O. Basir. Exchange strategies for multiple ant colony system. *Information Sciences*, 177(5), pp. 1248–1264, 2007.

[80] R. Eriksson, B. Olsson. On the behaviour of evolutionary global local hybrids with dynamic fitness functions. *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, LNCS 2439, pp. 13–22, Springer-Verlag, 2002.

[81] R. Eriksson, B. Olsson B. On the performance of evolutionary algorithms with life-time adaptation in dynamic fitness landscapes. *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pp. 1293–1300, IEEE Press, 2004.

[82] C.J. Eyckelhof, M. Snoek. Ant systems for a dynamic TSP: Ants caught in a traffic jam. *Proceedings of the 3rd International Workshop on Ant Algorithms*, LNCS 2463, pp. 88–99, Springer-Verlag, 2002.

[83] U. Faigle, W. Kern. Some convergence results for probabilistic tabu search. *ORSA Journal on Computing*, 4(1), pp. 32–37, 1992.

[84] S. Fenete, C. Solnon. Searching for maximum cliques with ant colony optimization. *Proceedings of 2002 EvoWorkshops on Applications of Evolutionary Computation*, G.R. Raidl, J.-A. Meyer, M. Middendorf, S. Cognoni, J.J.R. Cardalda, D.W. Corne, J. Cottlieb, A. Guillot, E. Hart, C.G. Johnson, E. Marchiori, editors, LNCS 2611, pp. 236–245, Springer-Verlag, 2003.

[85] T.A. Feo, M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6, pp. 109–133, 1995.

[86] S. Fidanova. Ant colony optimization for multiple knapsack problem and model bias, *Proceedings of the 3rd International Conference on Numerical Analysis and its Applications*, LNCS 3401, pp. 280–287, Springer-Verlag, 2005.

[87] S. Fidanova. Probabilistic model of ant colony optimization for multiple knapsack problem, *Proceedings of the 6th International Conference on Large-Scale Scientific Computing*, LNCS 4818, pp. 545–552, Springer-Verlag, 2008.

[88] N. Figlali, C. Ozkale, O. Engin, A. Figlali. Investigation of ant system parameter interactions by using design of experiments for job-shop scheduling problems, *Proceedings of the 35th International Conference on Computers and Industrial Engineering*, 58(2), pp. 538–559, 2009.

[89] L.J. Fogel, A.J. Owens, M.J. Walsh. *Artificial Intelligence through Simulated Evolution*, New York, John Wiley. 1966.

[90] H. Fu, B. Sendhoff, K. Tang, X. Yao. Characterizing environmental changes in robust optimisation over time. *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, pp. 551–558, IEEE Press, 2012.

[91] G. Fuellerer, K.F. Doerner, R.F. Hardl, M. Lori. Ant colony optimization for the two-dimensional loading vehicle routing problem, *Computers & Operations Research*, 36(3), pp. 655–673, Elsevier Science, 2009.

[92] C. Gagné, W.L. Price, M. Gravel. Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence dependent setup times. *Journal of the Operational Research Society*, 53(8), pp. 895–906, 2002.

[93] Y. Gajpal, P.L. Abad. Multi-ant colony system for a vehicle routing problem with backhauls. *European Journal of Operational Research*, 196(2009), pp. 102–117, Elsevier Science, 2008.

[94] Y. Gajpal, P.L. Abad. An ant colony system (ACS) for vehicle routing problem with simultaneous delivery and pickup, *Computers & Operations Research*, 36(12), pp. 3215–3223, Elsevier Science, 2009.

[95] Y. Gajpal, C. Rajendran, H. Ziegler. An ant colony algorithm for scheduling in flowshops with sequence- dependent setup times of jobs, *International Journal of Advanced Manufacturing Technology*, 30(5–6), pp. 416–424, 2006.

[96] L. Gao, Y. Zeng, A.G. Dong. An ant colony algorithm for solving max-cut problem, *Progress in Natural Science*, 18(9), pp. 1173–1178, Elsevier Science, 2008.

[97] L. M. Gambardella, M. Dorigo. Ant-Q: A reinforcement learning approach to the traveling salesman problem. *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, A. Prieditis, S. Russell, editors, pp. 252–260. Morgan Kaufmann, 1995.

[98] L. M. Gambardella, M. Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pp. 622–627, IEEE Press, 1996.

[99] L. M. Gambardella, M. Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3), pp. 237–255, 2000.

[100] L.M. Gambardella, É.D. Taillard, C. Agazzi. MACS-VRPTW: A multicolony ant colony system for vehicle routing problems with time windows. *New Ideas in Optimization*, pp. 63–76, 1999.

[101] L. M. Gambardella, E. D. Taillard, M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50(2), pp. 167–176, 1999.

[102] L.M. Gambardella, A. Rizzoli, F. Oliverio, N. Casagrande, A. Donati, R. Montemanni, E. Lucibello. Ant colony optimization for vehicle routing in advanced logistics systems, *Proceedings of the International Workshop on Modelling and Applied Simulation*, pp. 3–9, 2003.

[103] A. Gaspar, P. Collard. From GAs to artificial immune systems: Improving adaptation in time dependent optimization, *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, 3, pp. 1859–1866, IEEE Press, 1999.

[104] P.-P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez Belicositermes natalensis et Cubitermes sp. La théorie de la Stigmergie : Essai dinterprtation du comportement des termites constructeurs, *Insectes Sociaux*, 6, pp. 41–80, 1959.

[105] M.R. Garey, D.S. Johnson. *Computer and Intractability: A guide to the theory of NP-completeness*. San Francisco, Freeman, 1979

[106] M. Gendreau, F. Guertin, J.Y. Polvin, E. Tailard. Parallel tabu search for real-time vehicle routing and dispatching, *Transportation Science*, 33(4), pp. 381–390, 1999.

[107] J.J. Grefenestette. Genetic algorithms for changing environments. *Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature*, pp. 137–144, Elsevier Science, 1992.

[108] G. Ghiani, F. Guerriero, G. Laporte, R. Musmanno. Tabu search heuristics for the arc routing problem with intermediate facilities under capacity and length restrictions, *Journal of Mathematical Modelling and Algorithms*, 3(3), pp. 209–223, 2004.

[109] D. E. Goldberg, R. E. Smith. Nonstationary function optimization using genetic algorithms with dominance and diploidy, *Proceedings of the 2nd International Conference on Genetic Algorithms and their Application* , J. J. Grefenstette, editors, pp. 59-68, 1987.

[110] S. Goss, S. Aron, J.-L. Deneubourg, J.-M. Pasteels. Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76, pp. 579–581, 1989.

[111] M. Guntsch, M. Middendorf. Pheromone modification strategies for ant algorithms applied to dynamic TSP. *Proceedings of 2001 EvoWorkshops on Applications of Evolutionary Computation*, LNCS 2037, pp. 213–222, Springer-Verlag, 2001.

[112] M. Guntsch, M. Middendorf. A population based approach for ACO, *Proceedings of 2002 EvoWorkshops on Applications of Evolutionary Computation*, LNCS 2279, pp. 71–80, Springer-Verlag, 2002.

[113] M. Guntsch, M. Middendorf. Applying population based ACO to dynamic optimization problems. *Proceedings of the 3rd International Workshop on Ant Algorithms*, M. Dorigo, G.A. Di Caro, M Sampels, editors, LNCS 2463, pp. 111–122. Springer-Verlag, 2002.

[114] M. Guntsch, M. Middendorf, H. Schmeck. An ant colony optimization approach to dynamic TSP. *Proceedings of the 2001 International Conference on Genetic and Evolutionary Computation*, pp. 860–867, 2001.

[115] W.J. Gutjahr. A graph-based Ant System and its convergence, *Future Generation Computer Systems*, 16, pp. 873–888, 2000.

[116] T. Guo, Z. Michalewicz. Inver-over operator for the TSP. *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, LNCS 1498, pp. 803–812, Springer-Verlag, 1998.

[117] F. Glover. Tabu Search–Part 1. *ORSA Journal on Computing* 1(2), pp. 190–206. 1989.

[118] F. Glover. Tabu Search–Part 2. *ORSA Journal on Computing* 2(1), pp. 4–32. 1990.

[119] F. Glober, M. Laguna. *Tabu search.* Boston, Kluwer Academic Publishers. 1997.

[120] F. Glover, M. Laguna, R. Marti. Principles of tabu search. *Approximation Algorithms and Metaheuristics*, Chapman and Hall, 2005.

[121] W.E. Hart. A Theoretical Comparison of Evolutionary Algorithms and Simulated Annealing. *Proceedings of the 5th Annual Conference on Evolutionary Programming*, pp. 147–154, 1996.

[122] H. Handa, L. Chapman, X. Yao. Robust Salting Route Optimization Using Evolutionary Algorithms. *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, 1, pp. 497-517, IEEE Press, 2006.

[123] J. He, X. Yao, Drift analysis and average time complexity of evolutionary algorithms, *Artificial Intelligence*, 127(1), pp. 57–85, 2001.

[124] J. He, X. Yao. From an individual to a population: An analysis of the first hitting time of population-based evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5), pp. 495–511, 2002.

[125] M. Held, R.M. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM*, 10, pp. 196–210, 1968.

[126] K. Helsgaun. An Effective Implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1), pp. 106–130, 2000.

[127] W.D. Hills. Co-evolving parasites improve simulated evolution as an optimization procedure.*Physica D*, 42(1–3), pp. 228–234, 1990.

[128] B. Holldobler, E.O. Wilson. *The Ants.* Springer-Verlag, Berlin, Germany, 1990.

[129] J. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor, University of Michigan Press, 1975.

[130] X. Hu, R. Eberhart. Adaptive particle swarm optimisation: detection and response to dynamic systems, *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*, pp. 1666–1670, IEEE Press, 2002.

[131] A. Isaacs, V. Puttige, T. Ray, W. Smith, S. Anavatti. Development of a memetic algorithm for dynamic multi-objective optimization and its application for online neural network modelling of UAVs. *Proceedings of the 2008 IEEE International Joint Conference on Neural Networks*, pp. 548–554, IEEE Press, 2008.

[132] D.E. Jackson, F.L.W. Ratnieks Communication in ants. *Current Biology*, 16(15), pp. 570–574, Elsevier Science, 2006.

[133] P.K. Jain, P.K. Sharma. Solving job shop layout problem using ant colony optimization technique, *Proceedings of the 2005 IEEE International Conference on Systems, Man and Cybernetics*, 1–4, pp. 288–292, IEEE Press, 2005.

[134] T. Jansen, U. Schellbach. Theoretical analysis of a mutation-based evolutionary algorithm for a tracking problem in lattice, *Proceedings of the 2005 International Conference on Genetic and Evolutionary Computation Conference*, pp. 841–848, ACM Press, 2005.

[135] S. Janson, M. Middendorf. A hierarchical particle swarm optimizer for noisy and dynamic environments, *Genetic Programming and Evolvable Machines*, 7(4), pp. 329–354, 2006.

[136] E. Jen. Stable or robust? Whats the difference? *Complexity*, 8(3), pp. 12–18, 2003.

[137] Y. Jin, J. Branke. Evolutionary Optimization in Uncertain Environments - A survey. *IEEE Transactions on Evolutionary Computation*, 9(3), pp. 303–317, 2005.

[138] Y. Jin, B. Sendhoff. Constructing dynamic optimization test problems using the multi-objective optimization concept. *Proceedings of 2004 EvoWorkshops on Applications of Evolutionary Computation*, LNCS, 3005, pp. 525–536, 2004.

[139] D.S. Johnson, L.A. McGeoch. The travelling salesman problem: a case study in local optimization. *Local search in combinatorial optimization*, E.H.L Aarts, J.K. Lenstra (editors), pp. 215–310, John Wiley, 1997.

[140] R. Jovanovic, M. Tuba. An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem. *Applied Soft Computing*, 11(8), pp. 5360–5366, 2011.

[141] A. Kant, A. Sharma, S. Agarwal, S. Chandra. An ACO Approach to Job Scheduling in Grid environment, *Proceedings of the 1st International Conference on Swarm, Evolutionary and Memetic Computing*, LNCS 6466, pp. 286–295, Springer-Verlag, 2010.

[142] R.M. Karp. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, R.E. Miller, J.W. Thatcher (editors). New York, Plenum. pp. 85–103, 1972.

[143] H. Kawamura, M. Yamamoto, A. Ohuchi. Improved multiple ant colonies systems for traveling salesman problems. *Operations research / management science at work*, E. Kozan, A. Ohuchi, editors, pp. 41–59. Kluwer Academic, 2002.

[144] H. Kawamura, M. Yamamoto, K. Suzuki, A. Ohuchi. Multiple ant colonies algorithm based on colony level interactions. *IEICE Transactions, Fundamentals*, E83-A, pp. 371–379, 2000.

[145] L.J. Ke, Z.R. Feng, Z.G. Ren, X.L. Wei. An ant colony optimization approach for the multidimensional knapsack problem, *Journal of Heuristics*, 16(1), pp. 65–83, 2010.

[146] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi. Optimization by simulated annealing. *Science* 220(4598), pp. 671–680, 1983.

[147] P. Kilby, P. Prosser, P. Shaw. Dynamic VRPs: A study of scenarios, *Technical Report APES-06-1998*, University of Strathclyde, U.K., 1998.

[148] T. Koetzing, F. Neumann, H. Roglin, C. Witt. Theoretical analysis of two ACO approaches for the travelling salesman problem. *Swarm Intelligence*, 6(1), pp. 1–21, 2012.

[149] M. Kong, P. Tian. A new ant colony optimization applied for the multidimensional knapsack problem, *Proceedings of the 6th International Conference on Simulated Evolution and Learning*, LNCS 4247, pp. 142–149, Springer-Verlag, 2006.

[150] M. Kong, P. Tian, Y.C. Kao. A new ant colony optimization algorithm for the multidimensional knapsack problem, *Computers and Operations Research*, 35(8), pp. 2672–2683, 2008.

[151] P. Larrañaga, J.A. Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation.* Kluwer Academic Publishers, Boston, 2002.

[152] F.-X. Le Louarn, M. Gendrau, and J.-Y. Potvin. GENI ants for the traveling salesman problem. *INFORMS Fall 2000 Meeting*, pp. 5–8, 2000.

[153] G. Leguizamó n, Z. Michaelewicz. A new version of Ant System for subset problems. *Proceedings of the 1999 Congress on Evolutionary Computation*, pp. 1459–1464, IEEE Press, 1999.

[154] G. Leguizamó n, Z. Michaelewicz, M. Shütz. An Ant System for the maximum independent set problem. *Proceedings of the VI Argentinium Congress on Computer Science*, 2, pp. 1027–1040, 2001.

[155] J. Levine, F. Ducatelle. Ant colony optimisation and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society, Special Issue on Local Search*, 55(7), 2004.

[156] J. Lewis, E. Hart, G. Ritchie. A comparison of dominance mechanisms and simple mutation on non-stationary problems. *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, LNCS 1498, pp. 139–148, 1998.

[157] X. Li, J. Branke, T. Blackwell. Particle swarm with speciation and adaptation in a dynamic environment. *Proceedings of the 8th International Conference on Genetic and Evolutionary Computation*, pp. 51–58, ACM Press, 2006.

[158] X.Y. Li, P. Tian. An ant colony system for the open vehicle routing problem. *Proceedings of the 5th International Workshop on Ant Colony Optimization and Swarm Intelligence*, M. Dorigo, L.M. Gambardella, M. Martinoli, T. Stützle, editors, LNCS 4150, pp. 356–363, Springer-Verlag, 2006

[159] Y.-C. Liang, A. E. Smith. An Ant System approach to redundancy allocation. *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, pp. 1478–1484. IEEE Press, 1999.

[160] Y.-C. Liang, A. E. Smith. An ant colony optimization algorithm for the redundancy allocation problem. *IEEE Transactions on Reliability*, 53(3), pp. 417–423, IEEE Press, 2004.

[161] J. S. Liu. *Monte Carlo Strategies in Scientific Computing.* New York, Springer-Verlag, 2001.

[162] J.L. Liu. Rank-based ant colony optimization applied to dynamic travelling salesman problems, *Engineering Optimization*, 37(8), pp. 831–847, Taylor & Francis, 2005.

[163] A.Z. Liu, G.S. Deng, S.M. Shan. Mean-contribution ant system: An improved version of ant colony optimization for traveling salesman problem. *Proceedings of the 6th International Conference on Simulated Evolution and Learning*, LNCS, 4247, T.D. Wang, X. Li, S.H. Chen, X. Wang, H. Abbass, H. Iba, G. Chen, X. Yao, editors, pp. 489–496, Springer-Verlag, 2006.

[164] S. Lin, B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(2), pp. 498–516, 1973.

[165] T.D. Lin, C.C. Hsu, D.R. Chen, S.Y. Chiu. A new ant colony optimization algorithm with an escape mechanism for scheduling problems. *1st International Conference on Computational Collective Intelligence*, LNAI 5796, pp. 152–162, Springer-Verlag, 2009.

[166] D. Lohpetch, D. Corne. Multiobjective algorithms for financial trading: Multiobjective out-trades single-objective. *Proceedings of 2011 IEEE Congress on Evolutionary Computation*, pp. 192-199, IEEE Press, 2011.

[167] M. Lopez-Ibanez, T.D. Prasad, B. Paechter. Ant colony optimization for optimal control of pumps in water distribution networks, *Journal of Water Resources Planning and Management*, 134(4), pp. 337–346, 2008.

[168] S. Lorpunmanee, M.N. Sap, A.H. Abdullah, C. Chompoo-Inwai. Ant ant colony optimization for dynamic job scheduling in grid environment, *Proceedings of the World Academy of Science, Engineering and Technology*, 23, pp. 314–321, 2007.

[169] P. Lučič, D. Teodorovič. Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence. *Preprints of the TRISTAN IV Triennial Symposium on Transportation Analysis*, pp. 441–445, 2001

[170] H.R. Lourenoço, O. Martin, T. Stützle. Iterated local search. *Handbook of Metaheuristics*, F. Glover, G. Kochenberger, editors, 57, pp. 321–352, Kluwer Academic, 2002.

[171] H.R. Lourenoço, D. Serra. Adaptive search heuristic for the generalized assignment problem. *Math and Soft Computing*, 9(2–3), pp. 209–234, 2002.

[172] S. Mahfoud. Crowding and Preselection Revisited. *Proceedings of the 2nd International Conference of Parallel Problem Solving from Nature*, S. Manner, B. Manderick, editors, 2, pp. 27–36, Elsevier, 1992.

[173] V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problems. *INFORMS Journal on computing*, 11(4), pp. 358–369, 1999.

[174] V. Maniezzo, A. Carbonaro. An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, 16(8), pp. 927–935, 2000.

[175] V. Maniezzo and A. Colorni. The Ant System applied to the quadratic assignment problem. *IEEE Transactions on Knowledge and Data Engineering*, 11(5), pp. 769–778, 1999.

[176] V. Maniezzo, M. Milandri. An ant-based framework for very strongly constrained problems. *Proceedings of the 3rd International Workshop on Ant Algorithms*, M. Dorigo, G. Di Caro, M. Sampels, editors, LNCS 2463, pp. 222–227, Springer-Verlag, 2002.

[177] V. Maniezzo, A. Colorni, M. Dorigo. The ant system applied to the quadratic assignment problem. *Technical Report IRIDIA/94-28*, Universit Libre de Bruxelles, Belgium, 1994.

[178] G. Martinovic, D. Bajer. Elitist ant system with 2-opt local search for the travelling salesman problem. *Advances in Electrical and Computer Engineering*, 12(1), pp. 25–32, 2012.

[179] S. Marwaha, C. K. Tham, D. Srinivasan. Mobile agents based routing protocol for mobile ad hoc networks. *Proceedings of 2002 IEEE Conference on Global Telecommunications*, 1, pp. 163–167, 2002.

[180] M. Mavrovouniotis, S. Yang. A memetic ant colony optimization for the dynamic travelling salesman problem, *Soft Computing - A Fusion of Foundations, Methodologies and Applications*", 15(7), pp. 1405–1425, Springer-Verlag, 2011.

[181] M. Mavrovouniotis, S. Yang. An immigrants scheme based on environmental information for ant colony optimization for the dynamic travelling salesman problem, *Proceedings of the 10th International Conference on Artificial Evolution*, LNCS 7401, pp. 1–12, Springer-Verlag, 2011.

[182] M. Mavrovouniotis, S. Yang. A memetic algorithm based on ant colony optimization for the dynamic vehicle routing problem. A.S. Uyar, E. Ozcan and N. Urquhart, editors, *Automated Scheduling*, Springer-Verlang, 2012.

[183] N. Metropolis, A. Rosenbluth, M. Rosenbluth, M. Teller, E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6), pp. 1087–1092, 1953.

[184] D. Merkle, M. Middendorf. An ant algorithm with a new pheromone evaluation rule for total tardiness problems. *Proceedings of 2002 EvoWorkshops on Applications of Evolutionary Computation*, LNCS 1803, pp. 287–296, Springer-Verlag, 2000.

[185] D. Merkle, M. Middendorf. A new approach to solve permutation scheduling problems with ant colony optimization. *Proceedings of 2001 EvoWorkshops on Applications of Evolutionary Computation*, E.J. Boers, S. Cagnoni, J. Gottlieb, E. Hart, P.L. Lanzi, G. Raidl, R.E. Smith, H. Tijink, editors, LNCS 2037, pp. 213–222. Springer-Verlag, 2001.

[186] D. Merkle, M. Middendorf, H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4), pp. 333–346, 2002.

[187] R. Michel, M. Middendorf. An island model based ant system with lookahead for the shortest supersequence problem. *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, A. E. Eiben, T. Back, M. Schoenauer, H.-P. Schwefel, editors, pp. 692–701, Springer-Verlag, 1998.

[188] M. Middendorf, F. Reischle, H. Schmeck. Multi colony ant algorithms, *Journal of Heuristics*, 8, pp. 305–320, 2002.

[189] M. Middendorf, F. Reischle, H. Schmeck. Information exchange in multi colony ant algorithms, *Proceedings of the 3rd Workshop on Biologically Inspired Solutions to Parallel Processing Problems*, LNCS 1800, pp. 645–652, Springer-Verlag, 2000.

[190] M. Mourão, L. Amado. Heuristic method for a mixed capacitated arc routing a problem: A refuse collection application. *European Journal of Operational Research*, 160(1), pp. 139–153, 2005.

[191] M. Mouhoub, Z.J. Wang. Improving the ant colony optimization algorithm for the quadratic assignment problem, *Proceedings of the 2008 IEEE Congress on Evolutionary Computing*, pp. 250–257, IEEE Press, 2008.

[192] R.W. Morrison. Performance Measurement in Dynamic Environments. *Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation*, pp. 5–8, ACM Press, 2003.

[193] R.W. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*. Springer-Verlag, 2004.

[194] R.W. Morrison, K.A De Jong, A test problem generator for non-stationary environments. *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, pp. 2047–2053, IEEE Press, 1999.

[195] R. Montemanni, L. Gambardella, A. Rizzoli, A. Donati. A new algorithm for a dynamic vehicle routing problem based on ant colony system. *Proceedings of the 2nd International Workshop on Freight Transportation and Logistics*, pp. 27–30, 2003.

[196] R. Montemanni, L. Gambardella, A. Rizzoli, A. Donati. Ant colony system for a dynamic vehicle routing problem, *Journal of Combinatorial Optimization*, 10(4), pp. 327–343, 2005.

[197] F. Moyson, B. Manderick. *The collective behaviour of Ants : an example of self-organization in massive parallelism.* Artificial Intelligence Laboratory, 1988.

[198] F. Neumann, C. Witt. Runtime analysis of a simple ant colony optimization algorithm, *Algorithmica*, 54(2), pp. 243–255, 2009.

[199] C. Nothegger, A. Mayerm, A. Chwatal, G.R. Raidl. Solving the post enrolment course timetabling problem by ant colony optimization. *Annals of Operations Research*, 194(1), pp. 325–339, 2012.

[200] T. T. Nguyen, S. Yang, J. Branke. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation,* 6, pp. 1–24, Elsevier, 2012.

[201] T. Nguyen, X. Yao. Dynamic Time-Linkage Problems Revisited. *Proceedings of 2009 EvoWorkshops on Applications of Evolutionary Computation*, LNCS 5484, pp. 735–744, Springer-Verlag, 2009.

[202] F. Oppacher, M. Wineberg. The Shifting Balance Genetic Algorithm: Improving the GA in a Dynamic Environment, *Proceedings of the 1999 International Conference on Genetic and Evolutionary Computation*, 1, W. Banzhalf, editors, pp. 504–510, Morgan Kaufmann, 1999.

[203] A. Ostfeld, A. Tubaltzev. Ant colony optimization for least-cost design and operation of pumping water distribution systems, *Journal of Water Resources Planning and Management*, 134(2), pp. 107–118, 2008.

[204] M. Paletta, P. Herrero. A simulated annealing method to cover dynamic load balancing in grid environment, *International Symposium on Distributed Computing and Artificial Intelligence*, 50, J.M. Corchado, S. Rondriguez, J. Linas, J.M. Molina, editors, pp. 1–10, Springer-Verlag, 2010.

[205] H. Paessens. The savings algorithm for the vehicle routing problem. *European Journal of Operational Research*, 34, pp. 336–344, 1988.

[206] R.S. Parpinelli, H.S. Lopes, A.A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4), pp. 321–332, 2002.

[207] P. Pellegrini, T. Stützle, M. Birattari. A critical analysis of parameter adaptation in ant colony optimization, *Swarm Intelligence*, 6(1), pp. 23–48, Springer-Verlag, 2012.

[208] D.A.L. Piriyakumar, P. Levi. A new approach exploiting parallelism in ant colony optimization. *International Symposium on Micromechatronics and Human Sciences*, 7, pp. 237–243, 2002.

[209] V. Pillac, M. Gendreau, C. Guéret, A. L. Medaglia. A review of dynamic vehicle routing problems. *Technical Report, CIRRELT CIRRELT-2011-62*, 2011.

[210] C.M. Pintea, C. Chira, D. Dumitrescu, P.C. Pop. Sensitive ants in solving the generalized vehicle routing problem, *International Journal of Computers Communication and Control*, 6(4), pp. 734–741, 2011.

[211] B. Pfahringer. Multi-agent search for open shop scheduling: Adapting the Ant-Q formalism. *Technical Report, TR-96-09*, Austrian Research Institute for Artificial Intelligence, Vienna, 1996.

[212] M. Polacek, K. Doerner, R. Hartl, V. Maniezzo. A variable neighbourhood search for the capacitated arc routing problem with intermediate facilities, *Journal of Heuristics*, 14(5), pp. 405–423, 2008.

[213] A. Puris, R. Bello, Y. Martinez, A. Nowe. Two-stage ant colony optimization for solving the travelling salesman problem, *Proceedings of the 2nd International Conference on the Interplay Between Natural and Artificial Computation*, LNCS 4528, pp. 307–316, Springer-Verlag, 2007.

[214] W. Rand, R. Riolo. Measurements for understanding the behavior of the genetic algorithm in dynamic environments: a case study using the shaky ladder hyperplane-defined functions. *Proceedings of the 2005 International Conference on Genetic and Evolutionary computation Conference*, pp. 32–38, ACM Press, 2005.

[215] M. Rahoual, R. Hadji, V. Bachelet. Parallel Ant System for the set covering problem. *Proceedings of the 3rd International Workshop on Ant Algorithms*, M. Dorigo, G. Di Caro, M. Sampels, editors, LNCS 2463, pp. 262–267. Springer-Verlag, 2002.

[216] M. Reinmann, K. Doerner, R. Hartl. Insertion based ants for vehicle routing problems with backhauls and time windows. *Proceedings of the 3rd International Workshop on Ant Algorithms*, M. Dorigo, G. Di Caro, M. Sampels, editors, LNCS 2463, pp. 135–148. Springer-Verlag, 2002.

[217] M. Reinmann, K. Doerner, R. Hartl. Analyzing a unified ant system for the VRP and some of its variants. *Proceedings of 2003 EvoWorkshops on Applications of Evolutionary Computation*, G.R. Raidl, J.-A. Meyer, M. Middendorf, S. Cagnoni, J.J.R. Cardalda, D.W. Corne, J. Gottlieb, A. Guillot, E. Hart, C.G. Johnson, E. Marchiori, editors, LNCS 2611, pp. 300-310, Springer-Verlag, 2003,

[218] M. Reinmann, K. Doerner, R. Hartl. D-ants: Saving based ants divide and conquer the vehicle routing problem. *Computers and Operations Research*, 31(4), pp. 563–591, 2004.

[219] M. Reinmann, M. Stummer, K. Doerner. A saving based ant system for the vehicle routing problem. *Proceedings of the 2002 International Conference on Genetic and Evolutionary Computation*, W.B. Langdon, E. Cantú, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, N. Jonaska, editors, pp. 1317–1325, Morgan Kaufmann, 2002.

[220] H. Richter. Behavior of evolutionary algorithms in chaotically changing fitness landscapes. *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature*, LNCS 3242, pp. 111–120, Springer-Verlag, 2004.

[221] H. Richter. Evolutionary optimization in spatiotemporal fitness landscapes. *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature*, LNCS 4193, pp. 1–10, Springer-Verlag, 2006.

[222] H. Richter. Detecting change in dynamic fitness landscapes, *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, pp. 1613–1620, IEEE Press, 2009.

[223] H. Richter, S. Yang. Learning behavior in abstract memory schemes for dynamic optimization problems, *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 13(12), pp. 1163–1173, Springer-Verlag, 2009

[224] A. E. Rizzoli, R. Montemanni, E. Lucibello, and L. M Gambardella. Ant colony optimization for real-world vehicle routing problems - from theory to applications, *Journal of Swarm Intelligence*, 1(2), pp. 135–151, Springer-Verlag, 2007.

[225] P. Rohlfshagen, P. K. Lehre, X. Yao. Dynamic evolutionary optimisation: An analysis of frequency and magnitude of change, *Proceedings of the 2009 Genetic and Evolutionary Computation Conference*, pp. 1713–1720, ACM Press, 2009.

[226] P. Rohlfshagen, X. Yao. Dynamic combinatorial optimization problems: An analysis of the subset sum problem, *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 15(9), pp. 1723–1734, Springer-Verlag, 2011.

[227] E. Salari, K. Eshghi. An ACO algorithm for graph coloring problem, *Proceedings of the IEEE Congress on Computational Intelligence Methods and Applications*, pp. 179–183, IEEE Press, 2005.

[228] R. Schoonderwoerd, O. Holland, J. Bruten, L. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behaviour*, 5(2), pp. 169–207, 1996.

[229] R. Schoonderwoerd, O. Holland, J. Bruten. Ant-like agents for load balancing in telecommunications networks. *Proceedings of the 1st International Conference on Autonomous Agents*, pp. 209–216, ACM Press, 1997.

[230] M. Seo, D. Kim. Ant colony optimization with parametrised search space for the job shop scheduling problem, *International Journal of Production Research*, 48(4), pp. 1143–1154, Taylor & Travis, 2010.

[231] Y.Y. Shou. A bi-directional ant colony algorithm for resource constrained project scheduling, *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management*, 1–4, pp. 1027–1031, IEEE Press, 2007.

[232] S.J. Shyu, P.Y. Yin, M.T. Lin. An ant colony optimization algorithm for the minimum weight vertex cover problem, *Annals of Operations Research*, 131(1–4), pp. 283–304, Kluwer Academic, 2004.

[233] A. Shmygelska, H.H. Hoos. An improved ant colony optimization algorithm for the 2D HP protein folding protein, *Proceedings of the 16th Conference of the Canadian Society for Computational Studies of Intelligence*, LNAI 2671, pp. 400–417, Springer-Verlag, 2003.

[234] E. Sigel, B. Denby, S. Le Heárat-Mascle. Application of ant colony optimization to adaptive routing in a LEO telecommunications satellite network. *Annals of Telecommunications*, 57(5–6), pp. 520–539, 2002.

[235] A. Simõ s, E. Costa. An immune system-based genetic algorithm to deal with dynamic environments: diversity and memory. *Proceedings of the 6th International Conference on Neural Networks and Genetic Algorithms*, pp 168–174, 2003.

[236] F. Romeo, A. Sangiovanni-Vincentelli. A theoretical framework for simulated annealing, *Algorithmica*, 6(3), pp. 302–345, 1991.

[237] K. Socha. ACO for continuous and mixed-variable optimization. *Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence*, M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, T. Stutzle (editors), LNCS 3172, pp. 25–36, Springer-Verlag, 2004.

[238] K. Socha, J. Knowles, M. Sampels. A $\mathcal{MAX}-\mathcal{MIN}$ ant system for the university timetabling problem. *Proceedings of the 3rd International Workshop on Ant Algorithms*, M. Dorigo, G. Di Caro, M. Sampels, editors, LNCS 2463, pp. 1–13. Springer-Verlag, 2002.

[239] K. Socha, M. Sampels, M. Manfrin. Ant Algorithms for the University Course Timetabling Problem with regard to the state-of-the-art. *Proceedings of the 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization*, LNCS 2611, pp. 334345. Springer-Verlag, 2003.

[240] H.-P Schwefel. *Numerical optimization of computer models*, Chichester, UK, John Wiley, 1981.

[241] S. A. Stanhope, J. M. Daida. Genetic algorithm fitness dynamics in a changing environment, *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, 3, pp. 1851–1858, IEEE Press, 1999.

[242] T. Stützle. Parallelization strategies for ant colony optimization. *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, LNCS 1498, pp. 722–731, Springer-Verlag, 1998.

[243] T. Stützle. An Ant Approach to the Flow Shop Problem. *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing*, 3, pp. 1560–1564, 1998.

[244] T. Stützle, H.H. Hoos. The $\mathcal{MAX}-\mathcal{MIN}$ Ant System and local search for the travelling salesman problem. *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, T. Bäck, Z. Michalewicz, X. Yao, editors, pp. 309–314, IEEE Press, 1997.

[245] T. Stützle, H.H. Hoos. $\mathcal{MAX} - \mathcal{MIN}$ Ant System. *Future generation computer systems*, 16(8), pp. 889–914, 2000

[246] K. Tatsum, T. Tanino. A new rank-based ant system using different sensitive ants, *Symposium on Soft Computing and its Applications*, 4(5), pp. 1183–1193, 2008.

[247] E.-G. Talbi, O. Roux, C. Fonlupt, D. Robillard. Parallel ant colonies for combinatorial optimization problems. *Proceedings of the 1999 Workshops on Parallel and Distributed Processing*, 11, LNCS 1586, Springer-Verlag, 1999.

[248] E.-G. Talbi, O. Roux, C. Fonlupt, D. Robillard. Parallel ant colonies for the quadratic assignment problem. *Future Generation Computer Systems*, 17, pp. 441–449, 2001.

[249] D. Teodorovič. Bee colony optimization (BCO). *Innovations in Swarm Inteligence* C. P. Lim, L. C. Jain, S. Dehuri, editors, 65(215), pp. 39–60, Springer-Verlag, 2009.

[250] R. Tinos, S. Yang. An analysis of the XOR dynamic problem generator based on the dynamical system, *Proceedings of the 6th International Conference on Parallel Problems Solving from Nature*, LNCS 6238, pp. 274–238, 2010.

[251] K. Trojanowski, Z. Michalewicz. Searching for Optima in Non-stationary Environments, *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, 3, pp. 1843–1850, IEEE Press, 1999.

[252] S.P. Tseng, C.W. Tsai, M.C. Chiang, C.S Yang. A fast ant colony optimization for traveling salesman problem, *Proceedings of the 2010 IEEE Congress on Computational Intelligence*, pp. 1–6, IEEE Press, 2010.

[253] R. K. Ursem. Mutinational GA optimization techniques in dynamic environments, *Proceedings of the 2000 International Conference on Genetic and Evolutionary Computation Conference*, pp. 19–26, 2000.

[254] G. Theraulaz, E. Bonabeau. A brief history of stigmergy. *Artificial Life*, 5, 97–116, 1999

[255] R.J.M. Vaessens, E.H.L. Aarts, J.K. Lenstra. A local search template. *Technical Report COSOR 92-11*, Department of Mathematics and Computing Science, Eindhoven, 1995.

[256] F. Vavak, K. Jukes, T. C. Fogarty. Learning the local search range for genetic optimisation in nonstationary environments. *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, pp. 355–360, IEEE Press, 1997.

[257] F. Vavak, K. A. Jukes, T. C. Fogarty. Performance of a genetic algorithm with variable local search range relative to frequency for the environmental changes. *Proceedings of the 1998 International Conference on Genetic Programming*, Morgan Kaufmann, 1998.

[258] F. Vavak, T. C. Fogarty, K. Jukes. A genetic algorithm with variable range of local search for tracking changing environments. *Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature*, LNCS 1141, pp. pp 376–385, Springer-Verlag, 1996.

[259] A. Vogel, M. Fischer, H. Jaehn, T. Teich. Real-world shop floor scheduling by ant colony optimization. *Proceedings of the 3rd International Workshop on Ant Algorithms*, M. Dorigo, G. Di Caro, M. Sampels, editors, LNCS 2463, pp. 268–273. Springer-Verlag, 2002.

[260] C. Voudouris, E. Tsang. Guided local search. *European Journal of Operational Research*, 113(2), pp. 469-499, 1999.

[261] H. Wang, D. Wang, S. Yang. A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 13(8–9), pp. 763–780, 2009.

[262] K.P. Wang, L. Huang, C.G. Zhou, W. Pang. Particle swarm optimization for travelling salesman problem. *Proceedings of the 2nd International Conference on Machine Learning and Cybernetics*, 3, pp. 1583–1585, 1993.

[263] K. Weicker. Performance Measures for Dynamic Environments, *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, J. Merelo, P. Adamidis, H.G. Beyer, J. Fernández-Villacañas, H.P. Schwefel, editors, LNCS 2439, pp. 64–73, Springer-Verlag, 2002.

[264] K. Weicker. *Evolutionary Algorithms and Dynamic Optimization Problems.* Der Andere Verlag, Berlin, 2003.

[265] M. Wineberg, F. Oppacher. Enhancing the GAs ability to cope with dynamic environments, *Proceedings of the 2000 International Conference on Genetic and Evolutionary Computation*, pp. 3–10, 2000.

[266] L.N. Xing, Y.W. Chen, P. Wang, Q.S. Zhao, J. Xiong. Knowledge-based ant colony optimization for flexible job shop scheduling problems, *Applied Soft Computing*, 10(3), pp. 888–896, Elsevier Science, 2010.

[267] X.S. Xu, J. Ma, J.S. Lei. An improved ant colony optimization for the maximum clique problem, *Proceedings of the 3rd International Conference on Natural Computation*, pp. 766–770, IEEE Press, 2007.

[268] L. P. Wong, M. Y. H. Low, C. S. Chong, Bee colony optimization with local search for travelling salesman problem. *International Journal on Artificial Intelligence Tools*, 19, pp. 305–334, 2010.

[269] S. Wright. Evolution in mendelian populations, *Genetics*, 16(2), pp. 97–159, 1931.

[270] S. Wright. On the roles of directed and random changes in gene frequency in the genetics of populations, *International Journal of Organic Evolution*, 2(4), pp. 279–294, 1948.

[271] S. Yang. Non-stationary problem optimization using the primal-dual genetic algorithm. *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, pp. 2246–2253, IEEE Press 2003.

[272] S. Yang. Constructing dynamic test environments for genetic algorithms based on problem difficulty. *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, 2, pp. 1262–1269, 2004.

[273] S. Yang. On the design of diploid genetic algorithms for problem optimization in dynamic environments. *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pp. 1362–1369, IEEE Press, 2006.

[274] S. Yang. Genetic algorithms with elitism based immigrants for changing optimization problems. *Proceedings of 2007 EvoWorkshops on Applications of Evolutionary Computation*, M. Giacobini, editors, LNCS 4448, pp. 627–636, Springer-Verlag, 2007.

[275] S. Yang. Memory-based immigrants for genetic algorithms in dynamic environments, *Proceedings of the 2005 International Conference on Genetic and Evolutionary Computation Conference*, pp. 1115–1122, ACM, 2005.

[276] S. Yang. Memory-enhanced univariate marginal distribution algorithms for dynamic optimization problems. *Proceedings of the 2005 Congress on Evolutionary Computation*, 3, pp. 2560–2567, IEEE Press, 2005.

[277] S. Yang. Associative memory scheme for genetic algorithms in dynamic environments, *Proceedings of 2006 EvoWorkshops on Applications of Evolutionary Computation*, LNCS 3907, pp. 788-799, Springer-Verlag, 2006.

[278] S. Yang. Genetic algorithms with memory- and elitism-based immigrants in dynamic environments. *Evolutionary Computation*, 16(3), pp. 385–416, MIT Press, 2008.

[279] S. Yang, Y. Jiang, T. T. Nguyen. Metaheuristics for dynamic combinatorial optimization problems. *IMA Journal of Management Mathematics*, Oxford University Press, 2012.

[280] S. Yang, R. Tinos. A hybrid immigrants scheme for genetic algorithms in dynamic environments. *International Journal of Automation and Computing*, 4(3), 243–254, 2007.

[281] S. Yang, Y.-S Ong, Y. Jin. *Evolutionary computation in dynamic and Uucertain environments.* Computational Intelligence Series, 51, Springer-Verlag, 2007.

[282] S. Yang, X. Yao. Dual population-based incremental learning for problem optimization in dynamic environments. *Proceedings of the 7th Asia Pacific Symposium on Intelligent and Evolutionary Systems*, pp. 49–56, 2003.

[283] S. Yang, X. Yao. Population-based incremental learning with associative memory for dynamic environments. *IEEE Transactions on Evolutionary Computation*, 12(5), pp. 542–561, IEEE Press, 2008.

[284] J.Q. Yang, J.G. Yang, G.L. Chen. An improved ant colony system based on negative biased, *Proceedings of the International Conference on Advanced Measurement and Test*, Y.W. Wu, editors, pp. 439–440, pp. 558–562, 2010.

[285] X. Yao. Unpacking and understanding evolutionary algorithms. *Advances in Computational Intelligence*, LNCS 7311, pp. 60–76, Springer-Verlag, 2012.

[286] W.J. Yu, X.M Hu, J. Zhang, R.Z. Huang. Self-adaptive ant colony system for the travelling salesman problem. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 1399–1404, IEEE Press, 2009

[287] X. Yu, Y. Jin, K. Tang, X. Yao. Robust optimization over Time – A new perspective on dynamic optimization problems. *Proceedings of the 2010 IEEE Congress on Evolutionary Computation)*, pp. 3998–4003, 2010.

[288] X. Yu, K. Tang, X. Yao. An immigrants scheme based on environmental information for genetic algorithms in changing environments. *Proceedings of the 2008 IEEE Congress of Evolutionary Computation*, pp. 1141–1147, 2008.

[289] X. Yu, K. Tang, X. Yao. Immigrant schemes for evolutionary algorithms in dynamic environments: Adapting the replacement rate. *Science China Series F: Information Sciences*, 53(1), pp. 1–11, 2010.

[290] X. Yu, K. Tang, T. Chen, X. Yao. Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization. *Memetic Computing*, 1(1), pp. 3–24, 2009.

[291] E. L. Yu, P. N. Suganthan, Evolutionary programming with ensemble of explicit memories for dynamic optimization. *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, pp. 431–438, IEEE Press, 2009.

[292] Y.B. Yuan, K. Wang, L. Ding. A solution to resource-constrained project scheduling problem based on ant colony optimization algorithm. *Proceedings of the 9th International Conference on Hybrid Intelligent Systems*, 1, pp. 446–450, IEEE Press, 2009.

[293] A. Younes, O. Basir, O., P.A. Calamai. Benchmark generator for dynamic optimization, *Proceedings of the 3rd WSEAS International Conference on Soft Computing, Optimization, Simulation & Manufacturing Systems*, 2003.

[294] A. Younes, P. Calamai, O. Basir. Generalized benchmark generation for dynamic combinatorial problems, *Proceedings of the 2005 International Conference on Genetic and Evolutionary Computation Conference*, pp. 25–31, ACM Press, 2005.

[295] X.Y. Zhang, H.B. Duan, J.Q. Jin. DEACO: Hybrid ant colony optimization with differential evolution, *Proceedings of the 2008 IEEE Congress on Evolutionary Computation)*, pp. 921–927, IEEE Press, 2008.

[296] H.P. Zhang, M. Gen, S. Fujimura, K.W. Kim. Ant colony optimization approach for job-shop scheduling problem, *Proceedings of the 3rd International Conference on Information and Management Science*, 3, pp. 426–431, 2004.

[297] X.X. Zhang, L.X. Tang. A new hybrid ant colony optimization algorithm for the travelling salesman problem, *Proceedings of the 4th International Conference on Intelligent Computing*, LNCS 5227, pp. 148–155, Springer-Verlag, 2008.

[298] X.L. Zhou, J. Zhang, W.N. Cheng. A new pheromone design in ACS for solving JSP, *Proceedings of the 2007 IEEE Congress on Evolutionary Compuatation (CEC'07)*, pp. 1963–1969, IEEE Press, 2007.

[299] Y.M. Zhou, Q.S. Guo, R.W. Gan. Improved ACO algorithm for resrouce-constrained project scheduling problem, *International Conference on Artificial Intelligence and Computation Intelligence*, 3, pp. 358–365, IEEE Press, 2009.

[300] X. Zou, M. Wang, A. Zhou, B. Mckay. Evolutionary optimization based on chaotic sequence in dynamic environments, *Proceedings of the 2004 IEEE International Conference on Networking, Sensing and Control*, 2, pp. 1364–1369, IEEE Press, 2004.