CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
Diretoria de Pesquisa e Pós-Graduação
Programa de Pós-Graduação em Modelagem Matemática e Computacional

# Design configuration for MMAS algorithm applied for the Travelling Salesman Problem with Dynamic Demands

Tese de Doutorado apresentado ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, como parte dos requisitos necessários para a obtenção do título de Doutora em Modelagem Matemática e Computacional.

**Aluna** : Sabrina Moreira de Oliveira

**Orientador** : Prof. Dr. Sérgio Ricardo de Souza (CEFET-MG)

**Coorientadora** : Profa. Dra. Elizabeth Fialho Wanner (CEFET-MG)

**Coorientador** : Prof. Dr. Leonardo César Teonácio Bezerra (UFRN)

Belo Horizonte - MG
Junho de 2022

Oliveira, Sabrina Moreira de

O52d Design configuration for MMAS algorithm applied for the travelling salesman problem with dynamic demands / Sabrina Moreira de Oliveira. – 2022.

80 f.

Tese de doutorado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional.

Orientador: Sérgio Ricardo de Souza.

Coorientadores: Elizabeth Fialho Wanner e Leonardo César Teonácio Bezerra.

Tese (doutorado) – Centro Federal de Educação Tecnológica de Minas Gerais.

1. Formigas – Pesquisa – Teses. 2. Otimização combinatória – Teses. 3. Problema do caixeiro-viajante – Teses. 4. Controle automático – Teses. 5. Análise volumétrica – Teses. I. Souza, Sérgio Ricardo de. II. Wanner, Elizabeth Fialho. III. Bezerra, Leonardo César Teonácio. IV. Centro Federal de Educação Tecnológica de Minas Gerais. V. Título.

CDD 004.0151

Elaboração da ficha catalográfica pela bibliotecária Jane Marangon Duarte, CRB 6º 1592 / Cefet/MG

***"DESIGN CONFIGURATION FOR MMAS ALGORITHM APPLIED FOR THE TRAVELLING SALESMAN PROBLEM WITH DYNAMIC DEMANDS".***

Tese de Doutorado apresentada por **Sabrina Moreira de Oliveira**, em 24 de junho de 2022, ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, e aprovada pela banca examinadora constituída pelos professores:

Visto e permitida a impressão,

[illegible]
Profª. Drª. Elizabeth Fialho Wanner
Presidenta do Colegiadodo Programa de Pós-Graduação em
Modelagem Matemática e Computacional

To my daughters, as an example of resilience and effort.

# Acknowledgments

First, I would like to thank professors Sérgio Ricardo de Sousa, Elizabeth Wanner, and Leonardo Bezerra, who held my hands or even carried me all over these years when I needed it most. Professor Thomas Stützle, thanks for his kindness, patience and intellectual support. While studying at Iridia, I had the privilege to meet Prasanna Balaprakash and Marco Montes, friends who were also family during that time. I owe thanks for the privilege of having them in my life. Last but not least, I must thank my parents and sister, that never let me give up, and my husband who always supported me and believed in my dreams no matter what.

Be careful with your dreams, they might come true.

# Abstract

Ant colony optimization (ACO) algorithms were originally designed for static optimization problems where the input data is known *a priori* and is not allowed to undergo any change during their execution. Later, ACO memory proved to be effective in dealing with problems in which the benchmark is allowed to change in real-time without any prediction, that is, to solve Dynamic Combinatorial Optimization Problems (DCOPs). Among the main proposals of this kind, several adaptations of ACO procedures to improve information reuse can be identified in the Literature. In addition, the Population-Based ACO Algorithm (P-ACO) was designed specifically for DCOPs. Indeed, P-ACO drew the attention of the research community due to its ability to process faster pheromone information but the few studies that assessed the effectiveness of the procedures of ACO and specially P-ACO were not sufficient to achieve conclusions about the state of the art in ACO for Dynamic Optimization. In this work, we carried out an extensive experimental campaign to evaluate the most common adaptations of ACO main procedures identified in the literature, using the state-of-the-art ACO for static optimization as underlying algorithms, that is, the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ( $\mathcal{MM}$AS) and the most relevant ACO algorithm proposed for dynamic optimization, P-ACO. A variant of the traveling salesman problem, the TSP with dynamic demands (TSPDDs) was used as a test benchmark, similar to most investigations on ACO for combinatorial optimization. More importantly, a carefully designed experimental setup was adopted, which represented significant contributions to literature as follows. This is the first work that acknowledged that DCOPs required custom-configured parameter settings and also the first that used an automatic configuration tool for this task. In addition, we also showed how the hypervolume indicator could be used to evaluate the anytime behavior of algorithms for DCOPs. This new way of configuring and comparing the algorithm's performance was then applied between $\mathcal{MM}$AS and P-ACO. Comparing the performance of both algorithms showed that the first was able to consistently outperform the latter when a local search was adopted. Finally, we conducted an experimental investigation on the DCOP-specific components proposed for ACO, isolating local search, which is one of the most relevant procedures presented in some ACO algorithms, like $\mathcal{MM}$AS. Results showed that those components contributed very little to performance when algorithms were allowed to use local search but were remarkably effective in its absence. In fact, the used local search was not only feasible to be applied when dealing with dynamic components but also a must-use procedure, even for the TSPDDs where runtime was an issue to be coupled with. When a local search was applied, $\mathcal{MM}$AS was able to outperform P-ACO for a large part of the experimental setup we adopted.

**Keywords**: Ant Colony Optimization; Traveling Salesman Problem with Dynamic Demands; automatic configuration; hypervolume indicator.

# Resumo

Os algoritmos *Ant Colony Optimization* (ACO) foram desenvolvidos originalmente para problemas de otimização estáticos, nos quais os dados referentes ao problema tratado são conhecidos *a priori* e não são passíveis de mudanças durante a execução do algoritmo. Com o passar do tempo, o procedimento de memória pertencente ao algoritmo ACO provou ser efetivo para aplicação em problemas em que os dados são passíveis de alterações em tempo real, sem qualquer tipo de previsão, ou seja, se mostram eficientes também para serem aplicados em Problemas de Otimização Combinatória Dinâmicos (POCD). A partir de um estudo profundo das principais publicações sobre aplicação e comparação dos algoritmos ACO para POCD, diversos procedimentos de adaptação com o objetivo de aprimorar seu desempenho puderam ser encontrados na literatura, além da criação do algoritmo Population-Based Ant Colony Optimization( P-ACO), que foi desenvolvido especialmente para resolver POCD. De fato, P-ACO chamou a atenção da comunidade de pesquisa por sua habilidade de processar informações rapidamente. No entanto, os poucos estudos encontrados na literatura com o foco em analisar a efetividade dos procedimentos dos algoritmos ACO e, em especial, o algoritmo P-ACO, quando aplicados para a resolução de problemas dinâmicos, não foram suficientes para se alcançar conclusões sobre o estado da arte dos algoritmos ACO quando aplicados para essa nova classe de problemas. Assim, nessa pesquisa, é conduzida uma campanha extensiva de experimentos com o objetivo de estudar e avaliar as adaptações mais comuns dos principais procedimentos dos algoritmos identificados na literatura como estado da arte para resolução de problemas dinâmicos usando ACO, são eles: $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) e P-ACO. Uma variação direta do problema do caixeiro viajante clássico, o problema do caixeiro viajante com demanda dinâmica (PCVDD), foi escolhido como problema teste, assim como na maioria dos outros estudos em otimização combinatória apresentados na Literatura. Mais importante, é apresentada uma proposta de um setup de experimentos, o que representa uma das principais contribuições para Literatura, como justificado a seguir. Primeiramente, este é o primeiro trabalho que identifica a real necessidade do uso de uma configuração de parâmetros específica para problemas de otimização combinatória dinâmica e também o primeiro trabalho a utilizar uma ferramenta de configuração automática para um algoritmo aplicado para essa classe de problemas. A próxima contribuição desse trabalho se direciona a possibilidade do uso da técnica de hipervolume como indicador de desempenho dos algoritmos com o propósito de serem eficientes em qualquer horizonte de tempo em se tratando de problemas de otimização combinatória dinâmicos. Essa nova proposta de análise é aplicada para comparação de desempenho entre os algoritmos $\mathcal{MM}$AS e P-ACO. Ao se comparar os dois algoritmos $\mathcal{MM}$AS, foi capaz de obter melhores resultados em relaçãao P-ACO quando o procedimento de busca local é utilizado. Finalmente, experimentos são conduzidos com o objetivo de investigar componentes específicos dos algoritmos quando aplicados a problemas de otimização dinâmica com o isolamento do procedimento de busca local. Os resultados mostraram que, de fato, quando a busca local não é permitida, esses procedimentos são de grande importância. Por outro lado, quando o uso da busca local é permitido, a melhoria em desempenho causada por esses procedimentos é insignificante, ou até mesmo, inexistente. Outro fato importante é o melhor desempenho do algoritmo $\mathcal{MM}$AS em relação ao P-ACO com o uso de busca local e seu desempenho inferior quando a mesma não é permitida.

**Palavras-chaves**: Otimização por Colônia de Formigas, Problema do Caixeiro Viajante com Demanda Dinâmica, configuração automática, Indicador de hipervolume, procedimentos de atualização de feromônio.

# Contents

# List of Tables

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction

Optimization problems are often modeled as Combinatorial Optimization Problems (COPs), which involve finding values for a set of discrete variables related to a given objective function. A straightforward approach to defining the best solution for a COP could be the combination of all possible values in a search space and the choice of the best one. Unfortunately, this approach turns unfeasible for problems in which the number of possible values increases exponentially according to the increment of instance size. In addition to that fact, the run time of the algorithm applied to solve the problem and the time required to find an optimal combination values grow in the same proportion. Besides the issues mentioned above, when we talk about real-world optimization problems, other ones can arise depending on the problem tackled. An example is when *problem components* are allowed to change at runtime without any prediction. The COPs with these characteristics are known as Dynamic COPs (DCOPs).

Whenever the optimal solution for a COP cannot be efficiently obtained in practice, approximate algorithms such as heuristics and metaheuristics have been successfully applied to obtain near-optimal solutions. In addition, problems with dynamic environments impose some additional challenges to COPs. Such problems need to be re-optimized over time, on every problem change, to ensure not only feasibility but also the quality of the solutions.

Over the past decades, Ant Colony Optimization (ACO) (Dorigo e Stützle, 2004) has played a central role as a successful metaheuristic for COPs and as a potential technique to be extended for solving DCOPs. In ACO algorithms, artificial ants build solutions for the problem tackled stochastically, biased by (i) *a priori* problem-specific heuristic information, and (ii) pheromone information knowledge acquired over the algorithm run. This pheromone-based memory of ACO algorithms has proven well suited to static COPs where the input data do not change during the algorithm run. In this scenario, the knowledge acquired by ACO algorithms in the form of pheromone accumulates into a strong bias towards promising regions of solutions' search space.

In contrast to the highly advantageous use of a pheromone memory in static COPs, a dynamic scenario makes the long-term knowledge of ACO a good strategy for DCOPs. Because changes on the problem data are expected to only affect a portion of its original definition, ACO algorithms could reuse information learned before the changes to speed up the re-optimization cycle.

On the other hand, depending on how strong the problem data change is, this long-term knowledge might lead the algorithm to get stuck in a suboptimal solution in the search space. After an instance change, if the ACO algorithm has already converged to a specific region of the search space, this region can no longer be interesting or even feasible. As a consequence, the algorithm search might be delayed by the need to first forget part of its previous knowledge.

The most innovative ACO proposal targeting DCOPs is an algorithm meant to optimize the reuse of pheromone information after problem changes, which we refer to as *pheromone transfer*. In more detail, the Population-based ACO (P-ACO) (Guntsch, 2004) proposed a one-shot pheromone evaporation approach with the aid of a solution archive, in which the pheromone contribution from a given solution in the archive is removed whenever that solution leaves the archive.

Besides efficiency, this archive-based pheromone memory of P-ACO directly regulates the contribution timespan from solutions found before problem changes, and hence this algorithm quickly rose to become the ACO reference for DCOPs.

Another promising algorithm to be applied for solving DCOPs is the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) which is one of the best-performing ACO algorithms from static optimization. The use of a local search procedure and boundaries for pheromone values give the algorithm the balance between exploration and exploitation. Such balance is needed for tackling a problem in which the optimum solution is able to change on every problem instance change.

However innovative, the studies targeting ACO for DCOPs left important application aspects underlooked in the literature. When we talk about performance concerning different categories of COPs, parameters configuration must differ from each algorithm applied. Moreover, this configuration change must follow the addition or removal of algorithm's procedures, as same as the best way of solution quality measurement and comparison. Thus, the proper use of an ACO algorithm to real-world COPs must take into account the above mentioned aspects.

When we look at the literature concerning these main aspects, their importance is presented and recognized by the researches community. However, they have been partially applied and/or analyzed individually. In addition, each aspect mentioned has a direct impact on the application of the other one. Taking into account not all of them together could impose erroneous conclusions or under measure the performance of the algorithm tackled.

Given the fact that these aspects are crucial to define a properly ACO implementation, in this work we propose, (a) the proper configuration of parameters settings, (b) an adequate solution quality measurement for ACO algorithms, and (c) possible adaptations on ACO main procedures specifically for tackling DCOPs. Concerning what has been mentioned in (a) and (c), the efficacy of ACO algorithms depends on balancing heuristic and pheromone information. Not only on the *a priori* knowledge one has about the problem to be optimized, but also on the information learned during the execution of the algorithm. This balancing is regulated by numerical parameters and the algorithm's main procedures. Their proper setting requires significant knowledge of parameter configuration, ACO algorithms, how the procedures are applied and the problem instances one is dealing with.

Regarding (b), solution quality measurement in dynamic optimization, many measures evaluate algorithms based solely on the quality of the final solution they produce, completely disregarding the performance of the algorithm during different re-optimization cycles. Among the measures that assesses the behavior of algorithms over dynamic changes and solution quality development, some depend on a combination of other metrics, whereas others were proposed in the context of artificially designed test problems, where optimal solutions are known beforehand.

In order to proper use ACO algorithms when applied to DCOPs, in this work we propose a complete computational study regarding all aspects related to what has been emphasized in (a), (b) and (c).

Experiments were conducted on the Traveling Salesman Problem with Dynamic Demands (TSPDDs). We selected a variation of the Traveling Salesman Problem (TSP) because it is an extensively studied COP that has been used as test benchmark for many algorithms. In addition, ACO algorithms have been successfully applied to the TSP. In fact, we observe a similar pattern in the context of DCOPs, with many proposals (ACO or not) being firstly assessed on this problem. To assess algorithms on this variant of the TSP, we defined a complete experimental design that is applicable to DCOPs in general. The most important component of the proposed design is an effective way to configure parameter settings and evaluate performance for an *anytime behavior algorithm* by the use of the Hypervolume Indicator. The choice of this measure lies in the fact that it does not present the disadvantages observed for existing quality measurements applied to dynamic optimization, and has additional advantages demonstrated in the multi-objective optimization literature. Furthermore, we showed how it can be used to optimize the anytime behavior of dynamic optimization algorithms through *automatic parameter configuration.*

In this way, we proposed a bi-objective formulation of DCOPs, where runtime and solution quality are considered objectives to be simultaneously optimized. This idea is largely inspired by the use of the hypervolume measure to assess the *anytime behavior* of heuristic algorithms. In this work, we extended that concept to comprehend problem changes, enabling the comparison of different algorithms between consecutive changes (*environment-wise analysis*) or during their entire execution (*scenario-wise analysis*). In addition, our approach is based on an unary version of the hypervolume indicator, which makes it scalable as to the number of algorithms considered in the analysis. Our formulation is also scalable as to the number of objectives considered, meaning one can use it to assess the performance of dynamic multi-objective optimizers; yet, in this work we focused on the assessment of traditional DCOP optimizers.

We compared P-ACO and $\mathcal{MM}$AS to understand whether the novelties introduced in P-ACO determine a better performance. In particular when local search procedures and properly configured parameter settings are considered as factors of the investigation.

As a next stage of our investigation, we revisited some of the proposals to extend $\mathcal{MM}$AS to DCOPs assessing under our setup the improvements to $\mathcal{MM}$AS performance provided by those pheromone transfer mechanisms. More importantly, we chose $\mathcal{MM}$AS as ACO test benchmark to understand if those components contribute to the performance of effective ACO algorithms in general.

In summary, the contributions of this work have been divided into two main groups. The first one proposed the development of a complete experimental design and performance analysis for DCOPs. The latter was directly concerned about the application of ACO algorithms to COPs that were close to real-world optimization problems, and therefore, presents a detailed study on the influence of most relevant

ACO procedures when applied to DCOPs.

In order to achieve the goals that we proposed in this research, in Chapter 2 we introduced the existing dynamic variants of the TSP, highlighting the variant that we used as a benchmark in this work. In Chapter 3 we gave an overview of ACO algorithms and their adaptations to deal with DCOPs. After that, in Chapter 4, we targeted the performance assessment of DCOP algorithms, where we reviewed the two major metric categories, the ABC and hypervolume metrics from a critical perspective, and detailed both approaches in the context of dynamic optimization. In addition, we also discussed the parameter configuration as presented in Chapter 5 and Chapter 6. Finally, we dedicated Chapter 7 to describe our improvement proposals on $\mathcal{MM}$AS pheromone update and reported the computational study of these adaptations for dynamic optimization. Finally, conclusions and future work were discussed in Chapter 8.

# Chapter 2

# The Dynamic Travelling Salesman Problem (DTSP)

The Traveling Salesman problem (TSP) is one of the most widely studied *NP-hard* Combinatorial Optimization Problems. Despite its solving complexity, the TSP concepts are simple to implement and can be, most of the time, intuitive and straightforward (Gutin e Punnen, 2006; Monnot e Toulouse, 2014). Up to nowadays, these main characteristics have turned the TSP into one of the most desired problem classes used to introduce and compare algorithms performance (Mosayebi et al., 2021; Pina-Pardo et al., 2021). In addition, it is the problem in which Ant System (the first ACO algorithm presented in the Literature) was initially applied (Dorigo, 1992). Since then, the TSP has also been frequently used in ACO research to evaluate improvements on its main procedures (Dorigo e Stützle, 2004; Dorigo et al., 2011).

Without loss of generality, the classic TSP can be formulated as a COP where its main components are defined by a set of locations (customers) with certain distances connecting them. The goal is to find a closed tour of minimal length that visits each customer from this set exactly once.

A TSP instance can be represented by a fully connected graph $G = (V, E, d)$, $V$ being the set of $n = |V|$ vertices (representing the customers), $E$ being the set of edges that fully connects the vertices, and $d$ being a distance function that assigns to each edge $(i, j)$.

In the classic TSP, we assume that the distance function is symmetric, that is, we have $d_{ij} = d_{ji}$, meaning that the distance is the same whether one goes from $i$ to $j$ or in the opposite direction. Whenever $d_{ij} \neq d_{ji}$ then, the TSP is called as assimetric TSP (Gambardella e Dorigo, 1996; Schmitt et al., 2018, 2019).

Fomally, the TSP can be modelled as:

$$dec_{ij} \begin{cases} 1, & \text{if } (i,j) \text{ is covered in the tour} \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

in which $dec_{ij} \in \{0, 1\}$. Then, the problem objective function is defined as:

$$f(x) = \min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} dec_{ij}, \tag{2.2}$$

in which $n$ is the number of customers to be attended.

Different TSP approaches can be formulated from variants of its original definition. In the TSP with Time Windows (TSPTW), the customers must be attended in a specific time interval (time window) (Gendreau et al., 1998; López-Ibáñez e Blum, 2010). In the Probalistic TSP only an *a priori* subset of customers, taken randomly from the original benchmark set, is needed to be included in the closed tour (Gendreau et al., 1998). A review of the TSP main variations can be found at Gutin e Punnen (2006); Monnot e Toulouse (2014).

As mentioned in Chapter 1, in this research we chose the DTSP as a problem benchmark, more specifically, we tackled the TSP with dynamic demands. Due to its real-world nature characteristic, the way of modeling a DTSP is not strict. As opposite, it varies according to each researcher's aim. Therefore, the following contents of this Chapter are covered by the general description of the DTSP and the different ways of modeling a TSPDDs that have been presented in the Literature, so far.

## 2.1 The Dynamic Travelling Salesman Problem and main variations

The DTSP is a variation of the TSP that makes problem-solving more challenging, and, therefore, closer to real-world problems. As already mentioned, in the DTSP, the problem instance is allowed to change over time without any prediction.

Two main variants of the DTSP are presented in the literature. The first one considers that distances between the customers change over time, simulating the occurrence of traffic jams, accidents, or the change of weather conditions (Eyckelhof e Snoek, 2002; Melo et al., 2013; Mavrovouniotis e Yang, 2013b). The second considers demands as dynamic, in which the set of customers that need to be included in a tour can change over time, due to the cancellation of known visits or the appearance of new ones (Guntsch e Middendorf, 2001; Mavrovouniotis e Yang, 2011a, 2014a,b; Mavrovouniotis et al., 2014, 2015). In the literature, the first approach is named as TSP with Dynamic Traffic Jams (TSPDTJ), and the latter is named as TSP with Dynamic Demands (TSPDDs). As follows, we describe both approaches.

### 2.1.1 The Travelling Salesman Problem with Dynamic Traffic Jams (TSPDTJ)

In this formulation, for each pair of customers to be attended $i$ and $j$, there is a traffic factor $tf_{ij}$ assigned to the distance connecting them, that is, $d_{ij} = d_{ij} \times tf_{ij}$. During an algorithm run, the $tf_{ij}$ value represents the traffic at that moment (Mavrovouniotis e Yang, 2013b). In addition, two random numbers are associated with $tf_{ij}$ and $p_{tf}$. They are updated at each iteration. The first one denotes whether there is or not a traffic factor $r_{tf}$ assigned to the distance between $i$ and $j$. Distances with no traffic assigned have $r_{tf}$ set to 0. On the other hand, $r_{tf} = 1$, indicates traffic. The second one $p_{tf}$ is generated probabilistically and, in this case, values are generated according to $[LB, UB]$, in which $LB$ and $UB$ represent a lower and an upper bound, respectively. Values can vary from low, normal, or high. If $p_{tf}$

values are closer to $UB$ means high traffic, while for roads with low traffic, a higher probability is given to generate values of $p_{tf}$ closer to $LB$.

This class of DTSPs is denoted as random TSPDTJ. Another class is the one in which previously generated environments with their respective traffic jams are allowed to reappear. The dynamic changes occur with a cyclic pattern, that is, previous environments are guaranteed to appear again over the run time. Such environments are more realistic than scenarios that are randomly generated, for example, a rush in a traffic jam situation that might happen every day can be represented by an $tf_{ij}$ closer to $UB$ (Guntsch, 2004; Mavrovouniotis e Yang, 2013b).

Despite being classified as a dynamic problem, since some components of this formulation can be stochastic, turns out that the latter variation of TSPDTJs is not a dynamic problem class in a strict way. As presented in Guntsch (2004), probabilistic procedures can be used to handle how dynamic information is going to be generated as presented in the TSP with cyclic traffic jams. Despite being characterized as DTSP for some authors in the Literature, we do not agree that it is a dynamic problem in a *strict sense* as solutions can be reoptimized *a priori* given the probabilistically method used to generate the dynamic changes (Psaraftis et al., 2016). Thus, in this work, we put all our efforts into solving the DTSP which has all environments dynamically generated. We focused on solving the TSPDDs, which are presented as follows.

### 2.1.2 The Travelling Salesman Problem with Dynamic Demands

The TSPDDs is the DTSP problem class in which customers are allowed to be included or removed from a tour at any time during the algorithm run without any prediction.

Formally, the TSPDDs can be modelled as a sequence of graphs $G_s = (V_s, E_s)$, $s = 0, ..., S$, and two sequences of vertex sets $A_s$ and $D_s$, $s = 1, ..., S-1$. In particular, $A_s$ represents the set of new customers to be served and $D_s$ represents the set of deleted customers. We then have that $G_0 = (V_0, E_0)$ is the starting graph and each $V_s$ is obtained by $(V_{s-1} \bigcup A_s) \setminus D_s$ and $E_s = V_s \times V_s$.

Independently of which problem component is tackled as dynamic, when we move from the static to the DTSPs, two adaptations have to be faced. They are: (i) how to generate dynamic benchmarks and (ii) how to handle the dynamic changes over time. Different ways of generating dynamic benchmarks have been proposed in the literature depending on how re-optimization is done over the run and whether changes are periodic, continuous or cyclic. A review about several DOPs benchmark generators for both continuous and combinatorial DOPs is presented in Cruz et al. (2011); Nguyen et al. (2012a). In the following, the main adaptations applied on the TSPDDs are presented.

**Single insertion.** A way for generating dynamic environments is by what is called insertion scenarios, as presented in Guntsch e Middendorf (2001); Guntsch et al. (2001); Guntsch e Middendorf (2002); Guntsch (2004) in which a single customer is removed before the algorithm run and is latter reinserted. The opposite procedure is also feasible in which all customers are presented in the test

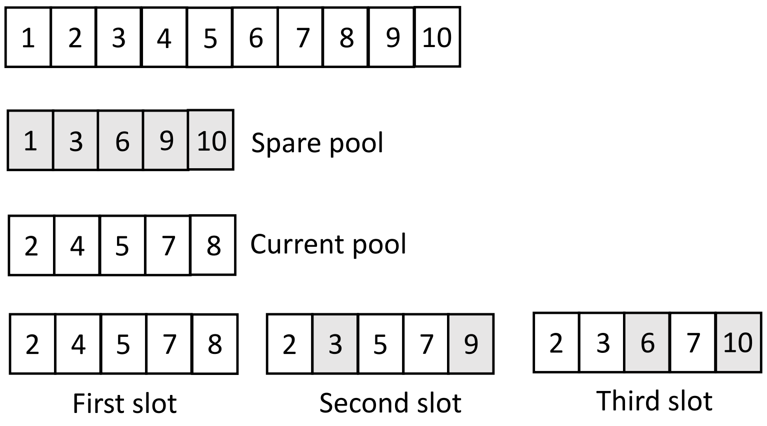


Figure 2.1: The TSP with dynamic demands modelled according to the pool insertion model

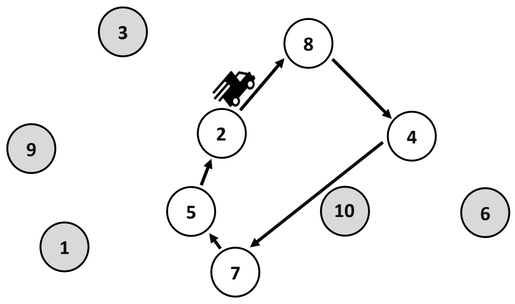
benchmark, and then, a single customer is removed, at any moment, over the runtime. In order to eliminate any bias concerning the position of the customer that has been removed or inserted, the algorithm is run over the removal and insertion of each customer that comprises the instance at a time. After that, the average of the solution quality is taken over all runs. For example, in Guntsch (2004) given the runtime $t$ as total number of iterations, for $t = 1500$, tests were run with insertion or removal of this single customer after 250 and 500 iterations.

The issue of using this methodology comes from the fact that, in the end, all components from the benchmark would have been removed/inserted. This fact, somehow, allows an *a priori* optimization, specially when the order of customers that has been removed/inserted is not random.

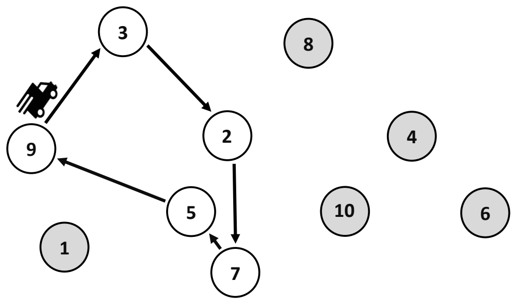
**Pool insertion.** Here, the set of customers from the actual problem instance is split into two sets called currentpool and sparepool. The former defines the current problem instance to be tackled and the latter defines the customers available to be switched with the ones in the currentpool. At specific moments, a fraction of the vertices are switched between currentpool and sparepool to define the new problem instance, regulated by a parameter $\xi \in [0, 1]$, here called *degree of dynamism*.

Another parameter that characterizes the instances is the *frequency of change*, which defines how often the instance in dynamic problem changes. While in Guntsch (2004); Mavrovouniotis e Yang (2011a, 2014a,b); Mavrovouniotis et al. (2014, 2015) the number of iterations/evaluations between changes is fixed, in this research the periodic re-optimization occurs according to runtime. Our rationale is that real-world dynamic problems are subject to asynchronous changes in time, with no regard to algorithmic concepts such as the number of iterations or the use of function evaluations. However, for simplicity, we keep the frequency of change fixed, i.e., environment changes happen synchronously producing $\epsilon$ time intervals evenly split. Besides modeling real-world problems more accurately, this change in the formulation has a direct impact on how algorithms should be engineered. More precisely, an algorithm that presents a high computational overhead at each iteration might have very few iterations to reoptimize solutions between changes, which is clearly an undesirable behavior.

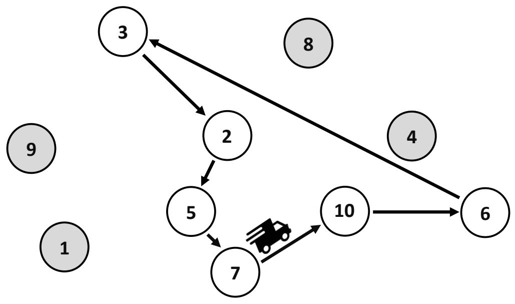
One step further, in order to create a more realistic benchmark, could be a random generation of $\xi$ values for each slot.



(a) Slot 01



(b) Slot 02



(c) Slot 03

Figure 2.2: Routes according to each slot presented at Figure 2.1.

Figure 2.1 illustrates the example of an instance with ten customers to be attended, that is $n = 10$. Thus, both currentpool and sparepool have size of $n/2$. The dynamic environment is defined by $\xi = 0.2$ and $\epsilon = 3$. In this way, the problem has three different slots and, at the end of the runtime defined for each slot, 20% of the customers are switched between the currentpool and sparepool. The customers added from the sparepool at each slot are shown in grey. In addition, Figure 2.2 shows the graphic representation of a feasible route for each slot. Figure 2.2a represents the feasible tour for the first slot, while the others 2.2b and 2.2c represent a feasible solution for the second and third tour, respectively. The customers that are not connected in order to form the round tour are the ones presented at the sparepool at that moment.

From the routes presented in Figure 2.2 (compare the routes presented in Figures 2.2a and 2.2b), it is easy to visualize the benchmark influence on the complexity of the route re-optimization over algorithm runtime. That is, benchmarks in which the customers are closer demand fewer efforts on the reoptimization process, while instances in which the location of customers are random, once the currentpool can have customers in many different locations as opposite as when customers are uniformly distributed.

# Chapter 3

# Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) is a search metaheuristic, inspired by the foraging behavior of real ants. More precisely, while foraging, some species of ants can leave on the ground a volatile substance called pheromone. This substance is reinforced according to the number of ants that have followed the same path. Over time, whenever two paths to the same food source are discovered, the shortest path connecting them will be the one with the higher amount of pheromone. This behavior lays in the fact that the pheromone trail from the shortest path will be more reinforced once ants will traverse it faster.

Due to this foraging behavior characteristic, the ACO metaheuristic can be applied straightforward for solving the classic TSP and its variations. In fact, it can be applied to any specific combinatorial optimization problem, in which candidate solutions are incrementally built and evaluated over a given objective function, under the consideration of adjusting or adding additional procedures such as problem-specific improvements. These procedures can be related to the way solutions are built, how pheromones are updated, or even the possibility of adding additional support, like the use of local search.

As a consequence, research on ACO main procedures for developing effective ACO variants of this COP class has been one of the most active research directions in the ACO research community. Some examples can be seen at Stützle e Hoos (1998); Stützle e Dorigo (1999); Guntsch (2004); Thalheim et al. (2008); Mavrovouniotis e Yang (2015); Schmitt et al. (2018); Mavrovouniotis et al. (2019).

An intuitive approach for building a solution for the classic TSP is to first randomly choose a vertex and then, at each step, go from the current vertex to the closest one that has not been visited yet. This solution construction ends when all vertices have been visited and the round trip is closed by returning to the initial vertex. Despite being intuitive, the solution quality from this *greedy* approach is most of the time the one with the highest values, especially when the instance size is increased. Other methodologies, like the branch and cut algorithm, can solve up to optimality small and medium symmetric TSP instances (1000 to 3000 customers) within feasible computation time. However, the computation time becomes unfeasible when increasing the instance size. Given this limitation, heuristic algorithms, in special heuristics with a stochastic local search procedure, became an attractive performance methodology for tackling this problem class (Hoos e Stützle, 2004). These methods are able to find optimal or close to optimal solutions with much lower computation time.

In this chapter, we focus on the application of ACO state-of-the-art algorithms for solving the TSP with dynamic demands (TSPDDs). They are: P-ACO and $\mathcal{MM}$AS. Firstly, we give the reader a general overview of the ACO metaheuristic. After that, P-ACO and $\mathcal{MM}$AS are detailed and analyzed. We give special attention to improvements that were developed and/or adapted on both algorithms when applied for the TSPDDs. The most important improvements can be mentioned as the changes in the pheromone update procedures and the use of local search. Concerning the latter, despite the computational cost that the use of local search brings to the algorithm run, in this research its use has been proved to be not just feasible, but desirable, or even necessary, when we apply ACO algorithms to DTSP benchmarks.

## 3.1 ACO Metaheuristic

As presented in Dorigo et al. (2011), a combinatorial optimization problem can be modeled by a tuple $(S, f, \Omega)$, in which:

- $S$ is the set of candidate solutions $s$, defined over a finite set of discrete decision variables $X$;
- $f : S \rightarrow \mathbb{R}$, is an objective function to be minimized;
- $\Omega$ is a set of constraints among the decision variables. Note that, depending on the problem tackled, it is allowed to be empty.

A decision variable $X_i \in X$, with $i = 1, ..., n$, is said to be instantiated when a value $v_i^j$ that belongs to its domain $D_i = \{v_i, ..., v_i^{|D_i|}\}$ is assigned to it. A solution $s \in S$ is called feasible if each decision variable has been instantiated satisfying all constraints in the set $\Omega$. Solving the optimization problem requires finding a solution $s^\star$ with minimun value of $f$ in a way that $f(s^\star) \leq \ f(s)\ \forall\ s \in \mathbb{S}$.

When ACO is applied to the TSP or other COP, after the initialization of the metaheuristic main parameters, three steps must be followed, respectively. First, the colony of $m$ ants builds candidate solutions according to $\Omega$ in an attempt to minimize $f$. After that, a specific procedure can be applied to improve the value of $f$ that has been defined. This procedure can vary according to the problem tackled or algorithm implemented and it is named as Deamon Actions procedure (Dorigo e Stützle, 2004). Finally, pheromone values are updated, increasing the influence of solution components that were associated with good solutions. These steps are repeated until the algorithm reaches the termination condition that can be a maximun run time or number of iterations. In most applications, these procedures are executed exactly in the aforementioned order. However, their use can vary according to the problem tackled and ACO variation adopted. The general outline of the ACO metaheuristic is presented in Algorithm 1.

As already mentioned in Chapter 2, the TSP is one of the most widely studied combinatorial optimization problems. From the COP point of view, in the construction graph $G = (C, L)$, $L$ represents the set of edges that connect all components from $C$. Analogously for the TSP, $C$ represents the set of customers to be attended, that is $C = V$, and, thus, $L = E$. The $\Omega$ values ensure the visibility of the tour that

**Algorithm 1** The pseudocode of ACO metaheuristic

```
Initialization
while termination condition is not met do
    Construct Solution
    Daemon Actions                                  ▷ Optional
    Pheromone Update
end while
```

has been built by the colony of ants. Because the tours must be represented by a complete graph, in this case, $\Omega$ ensures that only closed paths are allowed to be built without repetition of any customer.

Figure 1 presents the pseudocode of ACO metaheuristic. Next, we depict all the steps above mentioned when ACO is specifically applied to the TSP.

**Construct Solution.** For the classic TSP, a feasible solution is represented by a sequence of components $c_{ij} \in C$ and an instantiated decision variable represented by $X_i \leftarrow v_i^j$. A pheromone trail value $\tau_{ij}$ is associated to each solution component $c_{ij} \in C$. A solution construction starts from an initially empty partial solution $s^p$. At each construction step, $s^p$ is extended by appending to it a feasible solution component from the set of its feasible neighbors that satisfies the constraints in $\Omega$.

The choice of each $c_{ij}$ is guided by a probabilistic decision rule, which is biased by both the pheromone values $\tau_{ij}$ and the heuristic values $\eta_{ij}$ associated with $c_{ij}$. The pheromone values are represented by real numbers that are modified while running the algorithm. They reflect the learned desirability of choosing $j$ just after the inclusion of $i$ in the graph. The higher the pheromone value $\tau_{ij}$, the higher is the desirability of choosing $(i, j)$ as a solution component. Additionally, each edge has an associated heuristic value $\eta_{ij} = 1/d_{ij}$, which means the desirability of going from $i$ directly from $j$. The heuristic value is inversely proportional to the distance between $i$ and $j$. For the TSP, the value $\eta_{ij}$ is a measure of the heuristic desirability of having an edge $(i, j)$ as a component of a tour: the shorter the distance, the higher is the heuristic desirability. Note that, whenever $1/d_{ij} = 0$, $\eta_{ij}$ is set to a very small value. Both pheromone and heuristic information are represented by matrices of $\tau_{ij}$ and $\eta_{ij}$ values, respectively.

The exact rules for the probabilistic choice of solution components vary across different variants of ACO. As follows, we present the one applied in the first ACO algorithm, the Ant System algorithm (Dorigo et al., 1996), which works as an inspiration source procedure for other probabilistic choice variations:

$$p_{c_{ij}|s^p} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{c_{il} \in C(s^p)} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}, \tag{3.1}$$

in which $\tau_{ij}$ and $\eta_{ij}$ are, respectively, the pheromone trail value and the heuristic value associated with the component $c_{ij}$. The parameters $\alpha \geq 0$ and $\beta \geq 0$ determine the relative importance of pheromone versus heuristic information.

**Deamon Actions.** This procedure, although optional, is important for the state-of-the-art results of ACO algorithms, like the use of local search on $\mathcal{MM}$AS when applied to the TSP (Dorigo e Stützle, 2004). It allows the execution of problem-specific operations that cannot be performed by artificial ants. It is usually executed before the update of pheromone values in order to bias the ants' search toward high-quality solutions. In Section 3.2.4, the fundamentals of a local search procedure (ls) are presented. We give special concern to the *2-opt* ls, which is the one chosen to be applied as a Deamon Action procedure for $\mathcal{MM}$AS and P-ACO algorithms.

**Pheromone Update.** This procedure updates the pheromone trail values associated with each solution components $c_{ij}$. The modification of the pheromone trail values is performed in two steps. The first is the pheromone evaporation, which decreases the pheromone values of all components by a constant factor $\rho$ (called evaporation rate) to avoid premature convergence. The second is the pheromone deposit, which increases the pheromone trail values associated with components of a set of promising solutions $S_{upd}$. The general form of the pheromone update rule is as follows:

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \rho \cdot \sum_{s \in S_{upd} | c_{ij} \in s} F(s), \tag{3.2}$$

in which $\rho \in (0,1]$ and $f : S \rightarrow \mathbb{R}+$ is a function such that $f(s) < f(s') \implies F(s) \geq F(s'), \forall s \neq s \in S \cdot F(\cdot)$. Different definitions for the set $S_{upd}$ can be found in the literature. Two common choices are $S_{upd} = s_{bsf}$, and $S_{upd} = s_{ib}$, in which $s_{bsf}$ is the best-so-far solution, that is, the best solution found since the start of the algorithm, and $s_{ib}$ is the best solution of the current iteration. The specific implementation of the pheromone update mechanism differs across ACO variants as presented in Dorigo et al. (1991); Gambardella e Dorigo (1996); Dorigo e Gambardella (1997); Dorigo e Stützle (2004)

## 3.2 ACO algorithms applied to the TSPDDs

The high-level ACO metaheuristic description presented in Section 3.1 can vary depending on the algorithm tackled. For example, Ant System does not apply any Daemon Action procedure, and the Ant Colony System interleaves pheromone evaporation and solution construction. In the following, the variations on ACO main procedures are detailed for $\mathcal{MM}$AS and P-ACO. For both algorithms, the adaptations of the main algorithm procedures for tackling the TSP with dynamic demands are also presented.

### 3.2.1 $\mathcal{MM}$AS Ant System ($\mathcal{MM}$AS)

$\mathcal{MM}$AS is an improvement over the first ACO algorithm, Ant System (Stützle e Hoos, 1998; Stützle e Hoos, 2000; Dorigo et al., 2011). It is one of the most efficient algorithms for tackling COPs, in particular the classic TSP. If we compare $\mathcal{MM}$AS and Ant System, apart from the use of local search, the main adaptations of $\mathcal{MM}$AS concerns the management of the pheromone update. They are: (i) only one ant is allowed to deposit pheromone, (ii) the range of the pheromone trail values is bounded by two numerical parameters $\tau_{max}$ and $\tau_{min}$, and (iii) the pheromone trails are reinitialized every time the algorithm shows stagnation behavior.

As presented in Algorithm 2, $\tau_{max}$ and $\tau_{min}$ have their values initialized, in which $s^\star$ represents the solution quality of the optimal tour from the instance tackled and $a$ is a parameter (Stützle e Hoos, 2000). The objective is the imposing pheromone trail limits to the probability $p_{ij}$ of choosing a customer $j$ from a customer $i$. At the beginning of the algorithm run, all pheromone values are set to $\tau_{max}$.

At each iteration (it), when all ants have built their tour, only one ant is allowed to add pheromone in the pheromone matrix. It can be the *best-so-far ant* ($L^\star$), that is, the ant that built the tour with a minimum cost since the beginning of the algorithm run, or the *iteration-best ant* ($L^{'}$), which is represented by the ant that has to build the best tour in the current iteration.

The pheromone update rule of $\mathcal{MM}$AS is given below:

$$\tau_{ij}(it+1) = (1-\rho)\cdot\tau_{ij}(it+1) + \triangle\tau_{ij}^{best}, \tag{3.3}$$

in which $\rho \in [0,1]$ is the evaporation rate, $\triangle\tau_{ij}^{best} = 1/s^\star$ is a value that represents if edge $(i,j)$ belongs to the best tour ($s^\star$ or $s^{'}$). If edge $(i,j)$ belongs to the best tour, $s^\star$ is set to the length of the best tour or it is set to 0 otherwise. If the *best-so-far ant* is used, the $\tau_{ij}^{best} = 1/s^\star$ analougusly if the *best-of-iteration* tour lengh is used $\tau_{ij}^{best} = 1/s^{'}$.

**Algorithm 2** MMAS for Combinatorial Optimization Problems

```
ACO parameters Initialization
Inicialize MMAS parameters
$\tau_{max} \leftarrow 1/\rho \cdot s^\star$
$\tau_{min} \leftarrow \tau_{max}/a$
while termination condition is not met do
    while $i \in \{1, ..., m\} \leq m$ do
        Initialize selection set $S \mapsto \{0, 1, ..., n-1\}$
        Let $i$ construct solution $s_i$
    end while
    Determine the best solution of iteration $s^\star$
    Determine the best so far solution $s^\star$
    $\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \triangle\tau_{ij}^{best}$
    Apply Local Search                                        ▷ Optional
    if number of solutions without improvement is achieved then
        Reinicialize the pheromone trails
    end if
    if $\tau_{max}$ value is achieved then
        Reinicialize the pheromone trails
    end if
end while
```

The initialization of the pheromone values to an upper trail together with a small pheromone evaporation rate increases the exploration at the beginning of the run.

At the same time, the pheromone boundaries used $[\tau_{min}, \tau_{max}]$ avoid the algorithm stagnation behavior (when all ants build the same tour) over time. Every time the algorithm achieves the pre-defined limit of iterations without improvement, all pheromone values are reset to $\tau_{max}$.

Another important improvement procedure from $\mathcal{MM}$AS is the use of local search. Every time ants have completed their tour construction, solutions can be taken to their local optimum by the application of the local search procedure.

In the context of the TSP, many ACO algorithms such as $\mathcal{MM}$AS use nearest neighbor lists for speeding up solution construction, especially whenever a local search is applied. As presented in Oliveira et al. (2011b); Oliveira et al. (2017), for $\mathcal{MM}$AS almost 90% of its computation time is spent on the pheromone update since the evaporation affects all pheromone entries, which are quadratically many. When $\mathcal{MM}$AS is used with local search, the algorithm does not update the full pheromone matrix, but only the candidate's lists.

**$\mathcal{MM}$AS modifications for DCOP.** If we take the TSPDD as problem benchmark, the value of $s^\star$ will not be known anymore. Thus, at the beginning of the run, all pheromone values are set to $\tau_0$ with the same value assigned in Ant System, that is $\tau_0 = m/s^{nn}$, where $s^{nn}$ is the solution quality of the tour built by the nearest-neighbor heuristic. This setup of $\tau_0$ avoids the initialization of the pheromone matrix with values that are too low, that quickly bias the first tours generated, and at the same time are not too high to lead the algorithm taking many iterations until the pheromone evaporation is reduced enough to bias the ants. In addition, after every dynamic change, solutions that have been built so far may be not be feasible anymore. More precisely, whenever a DCOP moves from $G_s$ to $G_{s+1}$, the set of customers to be attended changes. Thus, the best solution found so far and also the nearest neighbor lists may change or even not be valid anymore. In this case, every time there is a change on $G_s$, the nearest neighbor lists are generated again for the new instance set defined. Note that the pheromone level of edges $V_{s-1} \setminus D_s \times A_s$ and $A_s \times A_s$, unless said otherwise, are initialized to $\tau_{max}$. The other edges (that is, all those in $V_{s-1} \setminus D_s \times D_{s-1} \setminus D_s$) keep the same values as before the move, and are used to identify the best-so-far solution by using the nearest-neighbor heuristic.

Besides being one of the state-of-the-art algorithms applied to TSP problems, the exploration characteristic of $\mathcal{MM}$AS at the beginning of the algorithm run, together with a boundary memory given from the use of $[\tau_{max}, \tau_{min}]$ have been gaining researchers' attention for expanding the $\mathcal{MM}$AS to problems with dynamic constraints. As mentioned above, after a problem change $G_{s+1}$, depending on how strong changes are, most of the tours may not be feasible anymore. As a consequence, the search space has also changed. In this way, after all, adjustments when moving from a benchmark to another, using an algorithm that has an explorative behavior at the beginning of the search together with the speed up procedure given from the use of nearest neighbors and local search turns $\mathcal{MM}$AS a state-of-the-art candidate algorithm also for DCOPs.

### 3.2.2 Population Based ACO (P-ACO)

P-ACO is an Ant Colony based algorithm proposed by Guntsch (Guntsch, 2004). As the majority of ACO algorithms, P-ACO was also designed for tackling the TSP, in special to deal with the TSP with components that are allowed to change over time without any prediction. Despite using the same solution construction procedure, P-ACO differs from other ACO algorithms in the way of updating the pheromone matrix. Instead of accumulating pheromone information provided by solutions constructed in every iteration, P-ACO maintains a solution archive and only solutions in this archive are reflected in the pheromone matrix.

The general form of the P-ACO pheromone matrix is as follows:

$$\tau_{ij}(it+1) = \tau_0 + \Delta \cdot |\{s \in P | (i,j) \in s\}|, \tag{3.4}$$

that is, the pheromone over edge $(i, j)$ at a given iteration $it + 1$ is equal to the initial pheromone deposit $\tau_0$ plus a deposit of $\Delta$, as many times as the number of occurrences of that edge in the solutions that belong to the archive.[1]

The pheromone upper bound $\tau_{max}$ is updated as:

$$\tau_{max} = \tau_0 + \Delta \cdot K, \tag{3.5}$$

and $\Delta$ is derived from

$$\Delta = \frac{\tau_{max} - \tau_0}{K}. \tag{3.6}$$

This pheromone update mechanism of P-ACO is typically faster than that of other ACO algorithms, which makes P-ACO a promising algorithm to be applied to dynamic COPs (Oliveira et al., 2011b; Oliveira et al., 2017). In addition, since only solutions in the archive influence the pheromone matrix, the pheromone matrix could be completely changed over time.

Different ways to manage when and which a solution is allowed to enter the archive have been proposed in the literature and are presented latter in this Section. The first and most intuitive procedure proposed is called *age-based strategy* (Guntsch, 2004). This procedure follows the *first in first out* (FIFO) queue behavior. The P-ACO pseudocode is presented below with the use of the *age-based strategy*.

Concretely, when a solution $s$ enters the solution archive $P$ (bounded to a maximum size $K$) being $k = 1$ its first element, $\Delta$ is added to the pheromone matrix corresponding to the solution component presented in a solution. When $|P| = K$, every new solution replaces the eldest solution in the solution archive (*age-based strategy*). When this replacement happens, the pheromone contribution of the components of the solution exiting $P$ are removed[2].

Overtime, depending on the problem and instance tackled the solution archive would have $k$ solutions presenting the same solution quality. As a consequence,

[1]Note that the pheromone over edge $(i, j)$ at a given iteration $it + 1$ does not depend on the pheromone over that edge on previous iterations and that the deposit does not depend on solution quality unless stated otherwise.

[2]Note that, when the algorithm starts, all entries of the pheromone matrix have an initial value of $\tau_0$, which has the same effect as the minimum pheromone trail limit in $\mathcal{MM}$AS (Stützle e Hoos, 2000), that is, pheromone values on the pheromone matrix concerned the components from $k = 1$ at $P$ are reduced to $-\Delta$

**Algorithm 3** P-ACO for Combinatorial Optimization Problems

```
while termination condition is not met do
    $\tau_{max} \leftarrow 1/\rho \cdot L^{\star}$
    $\tau_{min} \leftarrow \tau_{max}/a$
    Initialize pheromone values $\tau_{ij} \leftarrow \tau_0$
    Initialize the population $P \leftarrow 0$
    while $i \in \{1, ..., m\} \leq m$ do
        Initialize selection set $S \leftarrow \{0, 1, ..., n-1\}$
        Let $i$ construct solution $s_i$
    end while
    Determine the best solution of iteration $s'$
    Add $s'$ to population $P \leftarrow P \cup \{s'\}$
    for all $(i, j) \in s'$ do
        $\tau_{ij} \leftarrow \tau_{ij}(it + 1)$
    end for
    if $|P| > K$ then
        Remove solution $s^{k=0}$ from population $P$
        for all $(i, j) \in s^{k=1}$ do
            $\tau_{ij} \leftarrow \tau_{ij} - \Delta$
        end for
    end if
end while
```

the search exploitation gets intensified in a way it does not let the algorithm to improve solutions over time anymore. To manage this issue, some improvements have been proposed to keep balance between P-ACO exploration and exploitation over the algorithm run. These improvements are presented as follows.

#### 3.2.2.1 Population Management procedures

The P-ACO allows different ways of adapting the pheromone removal process according to the population of solutions. In the following, the procedures proposed by Guntsch in Guntsch (2004) are described. Figure 3.1 presents their graphic representation.

**Age-based strategy**. As already mentioned, the *age-based strategy* was the first population management strategy proposed by Guntsch (Guntsch, 2004). When proposed, it was the simplest method of managing the population of solutions following FIFO *(first in first out)* queue behavior.

**Quality-based strategy**. This strategy allows the *iteration-best solution* $s'$ to enter the solution archive only if its solution quality is better than the worst in $P$. Thus, if $s'$ is worse than all solutions in $P$, the solution archive remains unchanged. When $P$ is full, the new solution replaces the worst of $P$ at that moment and if the best solution of an iteration $s'$ is worse than all solutions in the population, the $P$ remains unchanged.

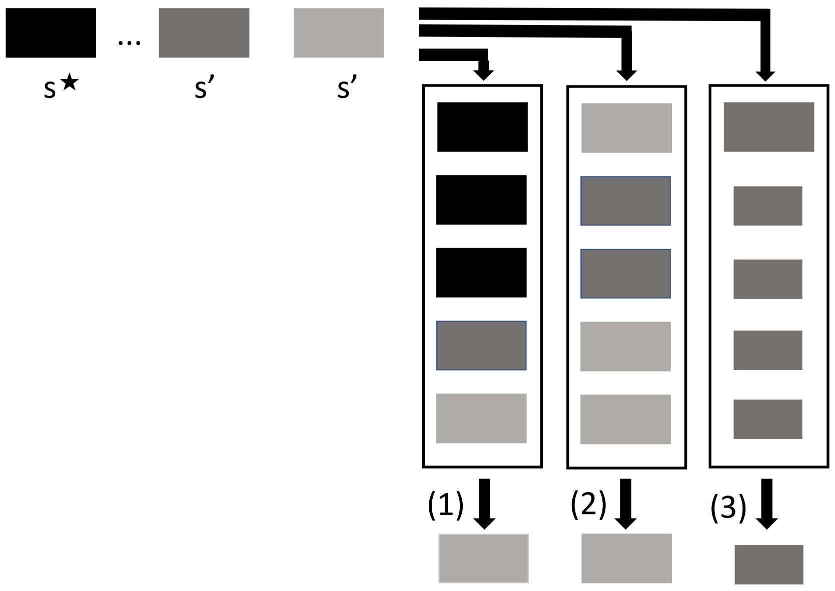


Figure 3.1: P-ACO Strategies: Graphical representation of P-ACO age (1), quality (2) and elitist (3) strategies, respectivelly.

**Elitist-based strategy**. This strategy keeps track of the best solutions found during a run. Each time a new best solution $e$ is found, the elitist update is applied. This means that, given a weight $w_e \in [0, 1]$, the best solution found so far receives a weight $w_e \cdot \tau_{max}$ and the other solutions of the population receive a weight $\Delta \cdot (1 - w_e)/(K - 1)$.

Figure 3.1 represents the dynamics of all strategies presented above. Each colored rectangle refers to a solution quality of $s$. Solutions in the solution archive are inside of bigger and blank rectangles. For each solution archive, $K$ was set to 5. Black rectangles represented the solution quality of *best so far solutions*, dark grey rectangles represented the solution quality with better performance than the ones in light grey. Thus, rectangles with the same colors have the same solution quality. The size of each rectangle represented the weight $w_e$ added to the solution quality for the *elitist-based strategy*. Figure 3.1(1) is the graphic representation of the *quality-based strategy*; Figure 3.1(2) represents the *age-based strategy*; while Figure 3.1(3) represents the *elitist-based strategy*.

In order to improve the performance of the strategies proposed, adaptations were implemented on *quality* and *age-based strategies*. They are:

**Probabilistic quality-based strategy**. When the *quality-based strategy* is used, after a certain number of iterations $P$ might consist of $K$ copies of the same solution. A way of minimizing this scenario was the introduction of a probabilistic component. $f(s)$ represents the solution quality of a solution $s$ in which lower values of $f(s)$ represent a higher solution quality. For $P = \{s_1, ..., s_{k+1}\}$, a probability distribution $pr$ is assigned over the solutions $s_i$, $i = 1, ..., K + 1$ in which:

$$pr_i = \frac{x_i}{\sum_{J=1}^{K+1} x_i}, \tag{3.7}$$

with

$$x_i = f(x_i) - \min_{j \in |1,k+1|} f(s_i) + avg(s), \tag{3.8}$$

and

$$avg(s) = \frac{1}{k+1} \sum_{J=1}^{K+1} f(s_i) - \min_{j=1,\ldots,k+1} f(s_i). \tag{3.9}$$

**Age & probabilistic-based strategy**. This strategy presents the possibility to combine the *age* and *probabilistic quality strategies*. Here, in the beginning of each run the *age-based strategy* is applied. However, the *probabilistic quality strategy* takes place just before the solution that would exceed the population archive limit is added. The combination of these strategies avoid a solution that has just been added into the archive to be immediately removed. At same time it guarantees that new solutions will influence the algorithms search for at least one iteration.

The same probability procedure as the one used on the *probabilistic quality-based strategy* is applied to the population of solutions before the $K$ limit exceeds. This action removes the possibility of having a solution that has just been added, to be removed before, at least, one iteration influence.

### 3.2.3 Reacting to Changes: Repair procedures for ACO algorithms

Whenever we apply an ACO algorithm to a DTSP, after each dynamic change the pheromone matrix needs to be adjusted as its objective is to serve as ants memory source. Over time, due to the dynamic changes on the problem components, some pheromone values may not make sense anymore.

As an example, consider the pool methodology that was applied in this research and presented in Section 2. Every time a customer $i$ was replaced at the current pool, the pheromone values connecting $i$ to any other customer $j$ did not make sense anymore. At the same time, a customer that had just been added to the current pool could present weekly connections to a promising $j$ as the pheromone values were set to $\tau_0$. The major consequence of this fact is the possibility of an algorithm getting stuck in a local optima area. Pheromone information from the outgoing customers might be so strong that would not allow the inclusion of any other newly added customers. As a consequence, these new customers could be added at the end of the tour, resulting in suboptimal solutions.

How much and where information is still exploitable and how much exploration would be needed just after the dynamic change to readapt the search has been the main issue faced by the ants' memory procedure for solving this problem class. To tackle this issue, several repair procedures on the pheromone update have been proposed and successfully tested in the literature (Guntsch e Middendorf, 2001; Guntsch, 2004; Mavrovouniotis e Yang, 2010, 2013b; Mavrovouniotis et al., 2014; Mavrovouniotis e Yang, 2014b). In the following we present the strategies used in this research that were proposed in Guntsch e Middendorf (2001); Guntsch (2004); Yang (2005) to enable the algorithms tested to explore the search behavior again, after the dynamic changes.

**Reset strategy.** In this strategy, pheromone information is removed on edges $V_{s-1}/D_s$ and are set according to a parameter $\gamma \in [0, 1]$.

**Restart strategy.** The restart strategy is similar to the strategy mentioned above. However, instead of removing the pheromone information on edges $V_{s-1}/D_s$, it is regulated by a forgetting parameter $\gamma \in [0, 1]$.

**$\eta$-strategy.** This strategy uses heuristic information to define the pheromone reset values instead of using an arbitrary parameter as presented in the reset strategy. Here, the heuristic distance $d^{\eta}_{ij}$ from a customer $i$ to any customer $j$ in the problem benchmark is defined by

$$d^{\eta}_{ij} = 1 - \frac{\eta_0}{\kappa \cdot \eta_{ij}}, \tag{3.10}$$

in which $\eta_0$ is a normalization parameter calculated by the average of all heuristic values as presented in Expression (3.11) and a parameter $\kappa \in (0, \infty)$. This measure is developed to define the maximun value of $d^{\eta}_{i}$ for all customers $j \in C$, limiting the minimun value to 0 as presented in Expression (3.13).

$$\eta_0 = 1 - \frac{1}{n \cdot (n-1)} \sum_{i} \sum_{i \neq j} \eta_{ij}. \tag{3.11}$$

$$d^{\eta}_{ij} = \max_{j \in C} d^{\eta}_{ij}. \tag{3.12}$$

$$\gamma_i = \max(0, d^{\eta}_{ij}). \tag{3.13}$$

**Immigrant Procedure.** The immigrant procedure was firstly introduced in the context of evolutionary algorithms (Yang, 2005, 2007, 2008; Xin Yu et al., 2008), and later extended to ACO algorithms (Mavrovouniotis e Yang, 2011a, 2012, 2013a,b, 2014b). In general, an immigrant can be either an ant or a solution that is moved from its original search context to another to promote diversification in a biased way. This is a default practice, when ACO algorithms are applied to DCOPs, as the pheromone information from many solution components created in past environments are reused in new ones. However, few proposals can be seen in the literature in which randomly modified solutions are used as immigrants (Mavrovouniotis e Yang, 2010), thus introducing a perturbation to the pheromone that is transferred across environments.

### 3.2.4 Local search and ACO

Constructive algorithms, like ACO, as presented in Algorithm 1, start from an initial empty solution and iteratively add solution components until the solution is complete. Although they are faster in comparison to approximate algorithms they present in general worst solution quality performance. This disadvantage can be surpassed by the use of local search methods. Without loss of generality, we can describe a Local Search (LS) operator for the TSP as a procedure that removes (or *unsign*) and inserts (or *sign*) nodes from a tour into such a position that improves the overall round trip cost. Solutions that are generated by using LS operator are named as candidate solutions.

Concerning the TSP, the literature shows that the use of local search, most of the time, significantly improves algorithms' performance. Starting from an initial solution $s$, a local search method repeatedly tries to improve it by local changes. As presented in Hoos e Stützle (2004), there are many variations of local search algorithms. In the case of the TSP, the stochastic local search (SLS) is a well-known high-performing LS algorithm and the one that is most successfully applied to this problem class. It is the one that makes use of randomized choices in generating or selecting candidate solutions for a given problem benchmark.

Before applying a LS procedure, we need to define what is called neighborhood structure. More precisely, we need to define for each current solution the set of possible solutions to which local search algorithms can move. The most efficient neighborhood structure applied to the TSP is by the use of whats is called $k$-*exchange* move. On the $k$-*exchange* method, $s$ and a candidate solution $s_{ngb}$ needs to be neighbors. That means that $s_{ngb}$ can be obtained from $s$ from deselection of a set of $k$ edges that are rewired into a complete solution by inserting a different set of edges. For the TSP, $k = 2$ and $k = 3$ are the most common choices. In this research, we focus on the use of $k = 2$, once it is the one that presents the best performance when $\mathcal{MM}$AS is applied. It is important to note that, when applied to ACO algorithms, $k$-*exchange* relations are called $k$-*opt*, because they build locally optimal solutions.

Figure 3.2 illustrates a 2-*opt* local search move. The most straightforward implementation of a 2-*opt* considers, in each step, all possible combinations of the two edges from a given candidate solution $s$, the number of ways the fragments can be reconnected into a different candidate solution different from $s$. For example, after deleting two edges $(7, 8)$ and $(2, 3)$, the way to rewrite the two partial tours is by introducing $(2, 7)$ and $(3, 8)$ with the reverse of one of the partial tours.

The use of local search presented an important role for ACO algorithms. However, the efficiency of its application depended on several factors, such as the initial solution, the neighborhood structure and the neighborhood search speed. In particular, the speed depended on move cost, that is, the time required for computing the difference of the cost value between couples of neighboring solutions.

When applied to ACO metaheuristic, after all ants completed their solution construction, solutions are taken to their local optimum by the application of the local search procedure. As described in Dorigo e Stützle (2004), because ACO's solution construction uses a different neighborhood than local search as presented in the beginning of this Section, and local search needs a high-quality starting solution, the

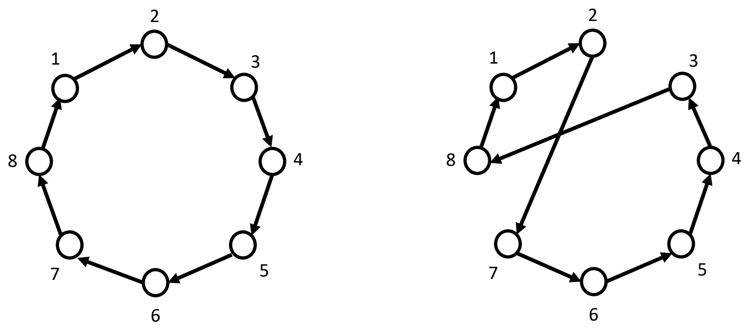


Figure 3.2: Graphical representation of a 2-*opt* local search. The left graph represents a solution $s = \{1, 2, 3, 4, 5, 6, 7, 8\}$ after a $s = \{1, 2, 7, 6, 5, 4, 3, 8\}$.

use of local search has a high probability to improve the solutions built by the ants. This is the case of $\mathcal{MM}$AS. The most important issue of using local search on COPs is that it is a very time-consuming metaheuristic, once run-time and solution quality are directly correlated.

To measure the impact between the time spent by the pheromone update procedure and the solution construction procedure, we run three ACO algorithms using a *profiling* tool.[3] Each algorithm was executed only once. P-ACO, $\mathcal{MM}$AS and Ant Colony System (ACS) were executed for the classic TSP. The comparison with ACS for the TSP was interesting because its pheromone update procedure also takes only $O(n)$ steps as P-ACO but with a larger constant hidden in the $O(n)$ notation.

The percentage of computation time for all functions related either to pheromone update or solution construction (including local search if executed), were summed and presented in Table 3.1.

As expected, P-ACO spent much less time for the pheromone update when compared to $\mathcal{MM}$AS. The amount of computation time saved was particularly impressive when compared to $\mathcal{MM}$AS on the TSP without a local search. For large instances, up to almost 90% of $\mathcal{MM}$AS's computation time was spent by the pheromone update (and not in generating solutions), due to the fact that the evaporation affects all pheromone entries, which were quadratically many (the big impact of the pheromone update was also due to the fact that the solution construction was of almost linear complexity because of the exploitation of candidate sets). For P-ACO, the computation time taken by the pheromone update was always a small percentage (around 5% in the discussed case). P-ACO's pheromone update was also much faster than ACS's one. Once local search was added, the advantage of the faster pheromone

[3] A profiler is a program used to check how much time is used by each function while executing the code.

| | **P-ACO** | | **ACS** | | **MMAS** | | **P-ACO+ls** | | **ACS+ls** | | **MMAS+ls** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance | *%cst* | *%ph* | *%cst* | *%ph* | *%cst* | *%ph* | *%cst* | *%ph* | *%cst* | *%ph* | *%cst* | *%ph* |
| d198 | 95.64 | **4.09** | 81.45 | **18.02** | 43.32 | **55.81** | 98.53 | **0.61** | 93.90 | **6.05** | 97.08 | **1.89** |
| eil101 | 93.63 | **5.87** | 83.02 | **16.86** | 47.14 | **52.56** | 97.20 | **0.85** | 95.50 | **4.17** | 97.40 | **1.24** |
| kroA200 | 94.77 | **4.86** | 77.77 | **22.02** | 42.70 | **56.98** | 97.91 | **0.85** | 95.43 | **4.26** | 97.44 | **1.47** |
| rd400 | 94.06 | **5.06** | 79.67 | **20.09** | 28.71 | **70.03** | 98.41 | **0.60** | 94.17 | **5.62** | 95.76 | **3.09** |
| pcb1173 | 94.53 | **5.02** | 77.49 | **22.34** | 11.11 | **88.21** | 97.41 | **1.57** | 93.15 | **6.08** | 95.75 | **3.86** |
| u2319 | 92.45 | **4.96** | 76.22 | **23.64** | 9.04 | **88.47** | 96.01 | **3.09** | 89.76 | **8.17** | 90.93 | **6.36** |

Table 3.1: *gprof* results summary - Percentage of computation time used for solution construction (including local search if executed), *%cst* and pheromone update, *%ph*. Two variants of P-ACO, ACS, MMAS (without local search, and with local search, *+ls*) were used and applied to the classic TSP. Note that in most cases the total computation time does not sum up to 100% due to other functions used by the code.

update by P-ACO was much reduced, due to two reasons. First, simply because relatively the local search took a significant amount of time; second, MMAS when used with local search did not update the full pheromone matrix, but only to the candidate sets. P-ACO's advantage with respect to computation time is nevertheless still clear in the TSP case, where tour length computation and construction is done in (almost) linear time.

In summary, the speedup of the pheromone update through P-ACO proved to be directly dependent by two factors: (i) the use of a local search, and (ii) the problem to which the algorithm is going to be applied, in particular, the complexity of the computation of the objective function. As a consequence, the pure speed up effect obtained by P-ACO's faster pheromone update depends strongly on these two features. From these experiments, P-ACO could be a good tool to be applied to problems in which local search does not perform well like the longest common subsequence problem (Blum et al., 2009) and the founder sequence reconstruction problem (Benedettini et al., 2010).

# Chapter 4

# Solution Quality Evaluation

Real-world optimization problems are often modeled as combinatorial optimization problems (COPs), which involve finding values for a set of discrete variables related to a given objective function. A straightforward approach to defining the best solution for a COP could be the combination of all possible values for each variable and the choice of the best set, that is, the one with the best solution quality for the given objective function. Unfortunately, this approach turns unfeasible for problems in which the number of possible solutions increases exponentially according to the increment of instance size. Due to that increment of instance size, not only the run time of the algorithm applied to solve the problem grows in the same proportion but also the time required to find an optimal solution. When the optimal solution cannot be efficiently obtained in practice, approximate algorithms such as heuristics and metaheuristics have been successfully applied to obtain near-optimal solutions. An example is the ACO metaheuristic presented in Section 3. Dynamic environments, in which instances are allowed to undergo some modifications over time without any prediction, as described in Section 2, impose some additional challenges to COPs since such problems need to be re-optimized overtime to ensure not only feasibility but also the quality of the solutions.

Some well-established evolutionary and swarm intelligence techniques have been tailored to solve dynamic COPs (DCOPs) (Eyckelhof e Snoek, 2002; Guntsch, 2004; Mavrovouniotis e Yang, 2011a; Nguyen et al., 2012a; Yang et al., 2013). However, to guarantee the effectiveness of the solution quality generated by an approximate algorithm, one has to execute it several times and solution quality of all executions should be compared in a way to prove whether their values are consistent or not. The literature is rich in examples of measures to assess the performance of algorithms applied to this context but many measures (i) evaluate algorithms based solely on the quality of the final solution they produce (Yang, 2005; Mavrovouniotis et al., 2017), completely disregarding the performance of the algorithm during different re-optimization cycles (*environments*); (ii) require several measures to be combined "to obtain knowledge about the behavior over dynamic changes and solution quality development" (Mavrovouniotis e Yang, 2011a; Guntsch, 2004), and/or; (iii) require the adaptation of measures that need *a priori* knowledge of the optimal solution for each problem change (Mavrovouniotis e Yang, 2010) since they were proposed in the context of artificially designed test problems in which optimal solutions are known beforehand.

In this chapter, we focus on the description of measures that have been proposed in the literature for tackling DCOPs. After that, aiming to overcome the issues discussed above, we proposed a bi-objective formulation of DCOPs, in which runtime and solution quality are considered objectives to be simultaneously optimized. This idea was largely inspired by Radulescu et al. (2013), who adopted the hypervolume measure to assess the *anytime behavior* of heuristic algorithms. Here, we extend that concept to comprehend problem changes, enabling the comparison of different algorithms between consecutive changes (*environment-wise analysis*) and during their entire execution (*scenario-wise analysis*). In addition, our approach is based on an unary version of the hypervolume indicator, which makes it scalable as to the number of algorithms considered in the analysis. Our formulation is also scalable as to the number of objectives considered, meaning one can use it to assess the performance of dynamic multi-objective optimizers. Yet, in this work, we focus on the assessment of traditional DCOP optimizers. All the experiments conducted to validate our proposal are presented later on.

## 4.1 Overview of metrics applied to DCOPs

The most common performance metrics for static COPs are based on the solution quality of the best solution found up to a given termination criterion (the final solution), averaged over a series of runs. In addition, when the optimal solution of a given instance is known *a priori*, as with the most commonly used TSP instances (Reinelt, 2008), it is possible to compute the relative percentage deviation (RPD) between the best solution found by an algorithm and its optimal solution. The average of the RPD over a series of runs is likely the most commonly adopted metric in static optimization (Hoos e Stützle, 2004). By contrast, for environment changes in dynamic COPs, a single value provided by the best solution quality average turns insufficient. Although different algorithms can display the same average value at the end of a run or just before a dynamic change, they can completely diverge on their search dynamics. Another important fact is that, due to the dynamic behavior of the instances tackled, as opposed to static TSP instances, the optimal solution is not known *a priori*. Thus, evaluating algorithms applied to DCOPs require re-evaluations of the quality of solutions, due to changes on the input data and/or to the optimal solution (Branke et al., 2005; Nguyen et al., 2012b).

Therefore, when we move from static to dynamic COPs, other evaluation criteria must arise to measure how fast the algorithm can react to dynamic changes or if the algorithm can maintain its performance over these changes. In fact, the goal of a dynamic optimizer is not only to retrieve a high-performing solution at the end of its run (or overall environment changes) but also to proceed with this retrieval efficiently. In a sense, each environment change forces the algorithm to restart its search, and each of these optimization cycles must be able to retrieve high-performing solutions consuming as few resources as possible.

Many performance assessment measures have been proposed in the dynamic optimization literature and can be classified as either *final quality-based* or *behavior-based*.

### 4.1.1 Solution quality based measures

The solution quality-based measures are based on the quality of the best solution found in each environment. For a scenario-wise analysis, performances across all environments are traditionally averaged. The major drawback with these approaches is the indifference to the search dynamics within environments. Specifically, a given algorithm may re-optimize solutions very quickly and still be considered equivalent to another that takes much longer to reach the same solution quality.

**Average best-of-generation**. The average best-of-generation is one of the most applied measures in dynamic environments. It was initially proposed for measuring the performance of evolutionary algorithms (Morrison, 2003; Alba e Sarasola, 2010a; Ben-Romdhane et al., 2013). The advantage of its methodology lies in the fact that it covers the entire optimization process. That is, it records the best solution quality over all generations $Gen$ over the total number of runs $R$. The best-of-generation formulation is presented bellow:

$$\overline{BOG} = \frac{1}{R}\frac{1}{Gen} \cdot \sum_{r=1}^{R} \sum_{gen=1}^{Gen} f(\overline{BOG}_{r,q}). \tag{4.1}$$

Besides the issue of being a metric that concerns only the solution quality development of an algorithm, another issue of applying $BOG$ concerns that it is not normalized.

**Offline performance**. When we tackle DCPs, the problem optimal solution $s^\star$ is not static anymore, following the environment changes. In this way, given the frequency of change $\zeta$, which defines how many environments the problem will tackle, the offline performance measures the average of the best so far solutions found at each slot $slt$ (Alba e Sarasola, 2010a). Equation 4.2 represents the formulation of the offlive performance measure:

$$off = \frac{1}{\zeta} \cdot \sum_{slt=1}^{\zeta} f(s^\star_{slt}). \tag{4.2}$$

The drawback of applying this method comes from the fact that the number of environment changes must be known *a priori*.

**Accuracy (Relative Error)**. The accuracy method, also named in the literature as *relative error*, was also a measure developed for static problems and then adapted to evaluate problems with dynamic constraints (Weicker, 2002). Given the fact that $BOG$ does not normalize solution quality, accuracy arises as a complimentary analysis of $BOG$. As presented in Equation 4.3:

$$accuracy = \frac{f(BOG) - MIN_t}{MAX_t - MIN_t}, \tag{4.3}$$

this measure enables us to find the best solution quality for each time interval, in which $MIN_t$ and $MAX_t$ are the worst and best know solution quality at a period $t$, respectively. The higher the accuracy value, the better the algorithm is.

### 4.1.2 Behavior based measures

The focus of the behavior-based measures relies on the measurement of algorithms' ability to quickly improve solution quality after the environment changes, or how much solution quality is lost between these changes.

Once it aims to analyze how an algorithm will adapt its search improvement over the environment changes, the behavior analysis of an algorithm requires the use of different approaches simultaneously or the use of interdependent measures.

In the following, we describe some of the most used behavior and how they are combined.

**Stability**. The stability is a measure of the algorithm's ability to recover its performance after each environment change. Thus, an algorithm can be considered stable if it can maintain its accuracy until the end of the run. Besides being a measure that does not allow to know whether the algorithm is improving well its solution quality, it is also accuracy dependent. That is, it requires the previous application of the accuracy measure to enable its use. The stability equation is presented as follows:

$$stability = max\{0, accuracy_t - accuracy_{t-1}\}. \tag{4.4}$$

$\beta_{\mathbf{Degradation}}$. The $\beta_{Degradation}$ is a very useful measure that was proposed to measure the algorithm's ability to track the instance moving optima. As presented in Alba e Sarasola (2010b), the performance of an algorithm tends to degrade the quality of solutions as the frequency of change increases. This behavior was also presented on the experiments conducted here specially when $\mathcal{MM}$AS with its default configuration is applied without local search for solving bigger instances sizes on scenarios with a higher degree of dynamism. Figure 4.3 illustrates this behavior.

To tackle this issue, the $\beta_{Degradation}$ measures how much of performance is lost over time. As presented in Equation (4.5):

$$u = \beta_{Degradation}\vec{ac} + \zeta, \tag{4.5}$$

given a sequence of accuracy values obtained at the end of a time period, in which $u$ represents an approximation to the algorithm overall accuracy vector $ac_sl$ of size $\zeta$, and degradation is the slope of the regression line and $\zeta$ is the number of periods in the dynamic problem, as shown in Equation (4.6):

$$ac_{sl} = \frac{1}{R} \cdot \sum_{r=1}^{R} f(s^{\star}_{slt,r}). \tag{4.6}$$

The accuracy for each period is obtained from the best solution found in period $slt$ averaged over all independent runs $R$. A positive degradation value indicates the algorithm keeps a good improvement and still provides good solutions: the bigger the improvement, the higher the slope value will be, being 1 its maximum value.

### 4.1.3 Area Between Curves measurement (ABC)

Contrarily to all other behavior metrics, the *area between curves* (ABC) (Alba e Sarasola, 2010a) does not require any assumptions about optimal solutions. This is illustrated in Figure 4.1, which depicts the performance fronts of two dynamic optimizers (left-most plots). In all plots, runtime is given on the $x$-axis, while solution quality is given on the $y$-axis (w.l.o.g. we consider a solution quality minimization problem). In more detail, the performance of a given algorithm $\Psi_A$ is represented as a set of points $\Phi_A = (\langle\phi_A^{t_1}, t_1\rangle, \langle\phi_A^{t_2}, t_2\rangle, \ldots, \langle\phi_A^{t_{max}}, t_{max}\rangle)$, in which $\phi_A^{t_i}$ is the solution quality of the best solution found by $\Psi_A$ up to time instant $t_i$. The ABC metric computes the area between the performance fronts (or curves) of two algorithms $\Psi_A$ and $\Psi_B$, illustrated in Figure 4.1 (right). In this figure, the solution quality of $\Psi_A$ is represented by black diamonds and $\Psi_B$ by white ones.

This metric can be mathematically modeled as the integral of the differences of $\Psi_A$ and $\Psi_B$ in the interval $[t_1, t_{max}]$ as:

$$abc^{\Psi_A} = \frac{1}{t_{max}} \int_1^{t_{max}} (\Psi_A)\, dt, \tag{4.7}$$

$$abc^{\Psi_B} = \frac{1}{t_{max}} \int_1^{t_{max}} (\Psi_B)\, dt, \tag{4.8}$$

$$ABC^{\Psi_A,\Psi_B} = \frac{1}{t_{max}} \int_1^{t_{max}} (\Psi_A - \Psi_B)\, dt. \tag{4.9}$$

If $ABC$ provides a positive result, the curve obtained by $\Psi_A$ has higher values when compared to $\Psi_B$. On the other hand, a negative value of $ABC$ means that curve $\Psi_B$ has higher values. The greater the distance between the two curves, the bigger the difference between algorithms performance.

In order to measure how better an algorithm is in comparison to the other, a *ratio* parameter is applied as follows:

$$\frac{abc^{\Psi_A}}{abc^{\Psi_B}} \geq 1 - \chi, \tag{4.10}$$

$$\epsilon - ratio = abc^{\Psi_A} \cdot \chi. \tag{4.11}$$

in which $\chi$ is a pre-defined parameter. In Alba e Sarasola (2010a), the authors used $\chi = 0.05$. Authors also consider the distance between the curves to be negligible, whether they are smaller than $\chi - ratio$. The term $\chi - ratio$ is used to name the maximum distance between some $abc^{\Psi_A}$ and other $abc^{\Psi}$.

In fact, this metric is a binary variant of a metric known in the context of multi-objective optimization as the *unary hypervolume indicator* (Radulescu et al., 2013), one of the best-established metrics for the performance analysis of multi-objective optimizers (Knowles et al., 2006). Specifically, the ABC metric is a particular case of the binary hypervolume metric where the reference point is only weakly dominated by the front assessed, and hence conclusions drawn from it cannot be guaranteed Pareto-compliant (Radulescu et al., 2013). More importantly, this poor choice of reference point (albeit implicit in the metric definition) means environment-final solutions may not be properly valued.

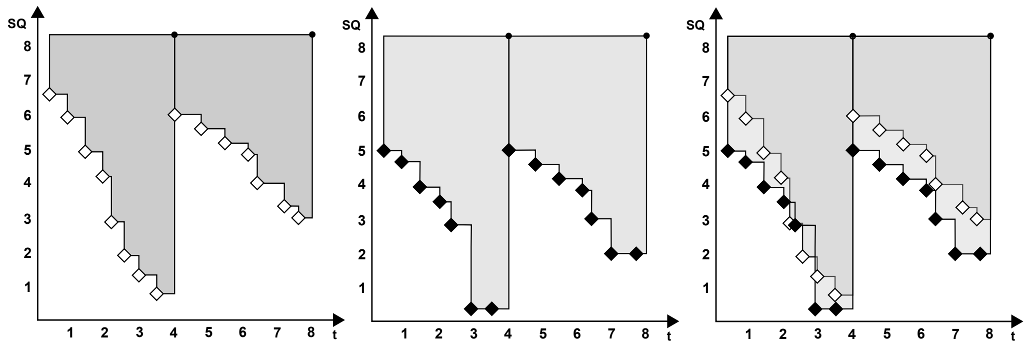


Figure 4.1: The hypervolume of two different sets of points (left and center), computed as a function of a bounding reference point. Given that points represent the performance of a dynamic optimizer, the hypervolume depicts its anytime behavior. In the dynamic optimization literature, the ABC metric (right) has been employed for a direct comparison of pairs of algorithms, measuring the difference between hypervolumes. Instead, in this work we compare multiple algorithms in an unary way, comparing the hypervolume values each algorithm achieves at a given run. We refer to the text for the reasons why we choose the unary hypervolume version over the binary one (ABC).

Notice that the ABC measure is loosely related to the multi-objective optimization performance assessment literature. Specifically, when assessing a single environment, this measure becomes close in spirit to the *binary hypervolume measure* (Zitzler et al., 2003). Yet, differently from the hypervolume, one cannot draw Pareto-compliant conclusions about the fronts being compared by the ABC, as we will later detail. In addition, this measure can only be applied to pairwise algorithm comparisons given its binary nature – the literature on binary measures for multi-objective optimization is clear that this is a non-scalable approach (Zitzler et al., 2003).

In the next section, we review another existing approach to a multi-objective formulation of algorithm performance, extend it to the context of dynamic optimization, and highlight the benefits of our approach over ABC.

The behavioral-based measures compare algorithms based on their search dynamics, providing more insights than final quality ones. Nonetheless, many of these measures may require the knowledge of the optimal solution to do so, or use an auxiliary measure that requires it. The only measure we identify without such restrictions is the *area between curves* (ABC, (Alba e Sarasola, 2010a)), which considers that the performance of algorithms are time-quality fronts and measure the area between these fronts for each environment. Aggregation for a scenario-wise analysis is traditionally done through an algebraic sum of the areas identified in each environment.

## 4.2 Assessing the anytime behavior of a dynamic optimizer: the hypervolume approach

In the context of static optimization, assessing the performance of an algorithm considering both resource consumption and solution quality is known as assessing its *anytime behavior* (Radulescu et al., 2013). When the resource consumption to be minimized is runtime, solution quality over time (SQT) plots can be used for a graphical analysis (Hoos e Stützle, 2004). For dynamic optimization, as presented above, several behavior-based metrics extend analytically the SQT to deal with multiple environments (Mori et al., 2001; Weicker, 2002; Rand e Riolo, 2005; Mavrovouniotis e Yang, 2011a; Cruz et al., 2011; Nguyen e Yao, 2012).

However, many such metrics have been proposed in a context in which artificial benchmark problems are used, having the knowledge of the optimal solutions for each environment at hand. In most real-world situations (and also here), such metrics cannot be applied as the optimal solution is in constant change across the different environments (see Chapter 2).

In Oliveira et al. (2019), we proposed the application of the hypervolume indicator to the context of dynamic optimization with minor adjustments. The high-level idea was to compute the hypervolume dominated by each algorithm in each environment separately and aggregate over environment-wise hypervolumes to draw overall conclusions. As illustrated in Figure 4.1 (right), we adopted environment-wise hypervolume measurements since, if the whole run of an algorithm were considered as a single front, most of the points depicting a given environment would be considered dominated by the best solution of the previous environment. To ensure that hypervolumes from multiple environments are comparable, we follow a two-stage approach. First, the values from both axes are globally scaled to ensure that both of them always contribute equally to the hypervolumes. Second, the reference point for a given environment is computed as an $x$-axis translation of a global reference point. As a consequence, solutions from each environment need to be evaluated in isolation, since the reference point of a given environment would intersect with the next environment.

The most important advantage of the hypervolume over the ABC metric was the preservation of the benefits of the original anytime behavior formulation. Specifically, given two algorithms $\Psi_A$ and $\Psi_B$, with their respective sets of performance-describing points $\Phi_A$ and $\Phi_B$, the formally proven properties of the hypervolume indicator ($I_H$) ensure that, if $\Psi_A$ presents a better anytime behavior than $\Psi_B$, then $I_H(\Phi_A) < I_H(\Phi_B)$. Alternatively, one can also say that, if $I_H(\Phi_A) < I_H(\Phi_B)$, then $\Psi_A$ cannot present a worse anytime behavior than $\Psi_B$. Additionally, using the unary hypervolume instead of its binary variants renders the analysis scalable w.r.t. the number of algorithms compared. Regarding an overall analysis, the benefits of our approach vary as a function of the aggregation method considered. A rank sum analysis indicated how often one algorithm reacted more efficiently to problem changes than others. In the most extreme case, an algorithm $\Psi_A$ presented larger hypervolumes than another algorithm $\Psi_B$ on all environments, so it is clear that $\Psi_A$ could not present worse anytime behavior than $\Psi_B$. Alternatively, one could also assess the average performance of an algorithm across environments. This flexibility of aggregation approaches was another improvement over the ABC metric, specially given that an algebraic sum implicitly embedded in the ABC metric provided less information than the alternatives discussed here.

In the following, we discussed the original anytime behavior formulation and its assessment through the hypervolume after that and extended it for the assessment of dynamic optimizers. We evaluated our proposal through the DTSPDDs and we discuss both experimental setup and results.

### 4.2.1 The hypervolume measure for the TSPDDs

The *anytime behavior* of an algorithm was defined by Radulescu et al. (2013) as the robustness of an algorithm to different stopping criteria. To compare different algorithms as to their anytime behavior, authors proposed that the original optimization problem under investigation be reformulated as a bi-objective problem, through the addition of a resource-minimizing objective. In the most traditional scenario, one wanted to optimize the *solution quality* of a target problem, and *runtime* was the resource which consumption was to be minimized. Under this formulation, the performance of an algorithm $\mathcal{A}_i$ was a nondominated front comprising points $\langle t_i, q_i \rangle$, i.e., the solution quality $q_i$ obtained at time $t_i$. Different algorithms could then be compared through the hypervolume they dominated, using a common reference point strictly dominated by all other points. Albeit simple, this approach is powerful in that (i) multiple algorithms can be simultaneously compared, and (ii) an algorithm that dominates a larger hypervolume cannot present a worse anytime behavior than one which dominates a smaller hypervolume (and vice-versa).

The application of the approach above to the context of dynamic optimization was straightforward when a single environment was considered. Yet, when problem changes are introduced, a few adjustments need to be made. To help illustrate these adjustments, Figure 4.1 depicted the performance fronts of two dynamic optimizers (left-most plots) and the comparison of their hypervolumes (right-most plot). In all plots, runtime is given on the $x$-axis, while solution quality is given on the $y$-axis (w.l.o.g. we consider a solution quality minimization problem). The first issue for computing a scenario-wise hypervolume illustrated in this figure was that, if the whole run of an algorithm was considered as a single front, most of the front depicting a given environment would be considered dominated by the best solution of the previous environment.

An alternative was to consider the environments separately and aggregate over environment-wise hypervolumes to draw scenario-wise conclusions. This approach required a second adjustment in the methodology, namely that reference points for each environment be computed as $x$-axis translations of the scenario-wise reference point. In more detail, to ensure all environment-wise hypervolumes were comparable (and hence could be aggregated), all reference points considered must present the same solution-quality coordinate. In addition, the time coordinate of each reference point was computed such that it could be strictly dominated by its environment front.[1] As we will later discuss, our case study presented fixed-duration environments and solution-quality ranges did not vary considerably across environments. Hence, we only applied a scenario-wise normalization to ensure both axes contributed equally to the hypervolumes. However, depending on the application considered, axes normalization for each environment might also be necessary.

Concerning environment-wise analysis, our approach preserved the benefits of the original anytime behavior formulation, which greatly improved over the ABC metric. Specifically, the ABC metric has been a particular case of the binary hypervolume

[1] In practice, this required isolating fronts from each environment before computing hypervolumes, since the reference point of a given environment might intersect with the next environment.

metric where the reference point is only weakly dominated by the front assessed, and hence conclusions drawn from it could not be guaranteed Pareto-compliant. More importantly, this poor choice of reference point (albeit implicit in the metric definition) meant environment-final solutions were not properly valued. By contrast, as long as standard guidelines about the hypervolume were followed, these solutions were guaranteed to be properly valued.[2]

Regarding a scenario-wise analysis, the benefits of our approach vary as a function of the aggregation method considered. If one uses a rank sum analysis, one could understand how often one algorithm reacted more efficiently to problem changes than others. More importantly, if an algorithm $\mathcal{A}_1$ presented larger hypervolumes than another algorithm $\mathcal{A}_2$ on all environments, one could be sure that $\mathcal{A}_1$ could present worse anytime behavior than $\mathcal{A}_2$. Conversely, if one was more interested in average performance, it was straightforward to compare algorithms based on the average of the hypervolumes computed for each environment. This flexibility of aggregation approaches was another improvement over the ABC measure, specially given that an algebraic sum implicitly embedded in the ABC measure provided less information than the alternatives discussed here.

Next, we present a case study where we empirically evaluated our proposed approach which was published in Oliveira et al. (2019).

## 4.3 Applying an anytime behavior of a dynamic optimizer for the TSDDs

The formulation and measure we proposed to employ in Oliveira et al. (2019) was general enough to assess the performance of any dynamic optimization algorithm on any given DOP. In that research, we conducted an experimental evaluation using the DTSPDDs as test benchmark and the P-ACO as target algorithm. Next, we detail the experimental setup we adopted, and later proceed to a discussion of the results observed.

### 4.3.1 Experimental setup

We remark that all adaptations of ACO algorithms for the DTSP presented in this research were implemented and executed on top of the ACOTSP software package available at `http://www.aco-metaheuristic.org/aco-code/`. In the same way, all experiments conducted here have been run on AMD Opteron CPUs with 12MB cache and 16 GB of RAM running under Cluster Rocks Linux. The algorithms studied were coded in **C** and compiled with **gcc** version 4.1.2.

In this work, we considered three P-ACO variants, and their parameter configurations are depicted in Table 4.1, in which $m$ is the number of ants, $\alpha$ and $\beta$ respectively regulate the importance of pheromone and heuristic information, $\tau_{max}$ is the maximum pheromone deposit for a given edge, and $K$ is the solution archive size (Oliveira et al., 2019). The first two variants, dubbed static- and dynamic-default, differ only as

[2]One could argue that an application might require a custom importance distribution for the different stages of the run. This could be achieved through the *weighted hypervolume measure*, as proposed in Radulescu et al. (2013).

Table 4.1: Default parameters used in the literature for P-ACO. *Dynamic-default + ls* settings are not given because no study has yet investigated this setup.

| Settings | $m$ | $\alpha$ | $\beta$ | $\tau_{max}$ | $K$ |
|---|---|---|---|---|---|
| *static-default* | $n/4$ | 1 | 2 | 3 | 25 |
| *dynamic-default* | 50 | 1 | 5 | 3 | 3 |
| *static-default + ls* | 25 | 1 | 2 | 3 | 1 |

to their parameter configuration, using the values traditionally employed for solving static (Oliveira et al., 2011a) and dynamic problems (Mavrovouniotis e Yang, 2011a), respectively. The third variant, dubbed static-default + ls, differs from the previous two since it is the only variant that adopts a local search (LS) procedure. We remark that this configuration had previously been only applied to static problems, due to the expected computational overhead posed by local search procedures. Yet, we include this variant in our study given the important role that local search plays for ACO effectiveness on static COPs, providing algorithms a means to locally explore a neighborhood in the search space. Specifically, we adopt the 2-opt neighborhood operator with a first improvement pivoting rule.

We adopted two sets of TSPDD instances, namely TSPLIB instances rl1323, u1817, rl1889, u2152, and pr2392 (Reinelt, 2008), and 3 subsets of 5 random uniform Euclidean (RUE) instances each, with sizes ranging from 3000 to 4000, created though the portgen generator from the 8th DIMACS implementation challenge (http://dimacs.rutgers.edu/archive/Challenges/TSP/, 2018). Each instance is further parameterized by the degree of dynamism ($\xi \in \{20\%, 40\%, 80\%\}$) and frequency of change ($f \in \{2, 10\}$). Algorithms were allowed a maximum runtime of 1 000 seconds, which meant environments last 500 seconds when $f = 2$ and 100s when $f = 10$. To account for variance, each algorithm was run 20 times on each instance configuration, and results reported were averages of those runs.

The hypervolume computation required a few pre-processing steps, as follows. First, a solution quality range was computed for each instance, and the runs of all algorithms were normalized so that the worst solution quality value ever found by an algorithm for a given instance corresponded to 0, and the best to 1. Time coordinates were normalized in the $[0, 1\,000]$ range, and hence both objectives contributed equally to the hypervolume. As previously discussed, the reference points for each environment were computed as $x$-axis translations of the global reference point $(1.1, 1.1)$. We ensured environment-specific reference points equally valued extreme solutions by using the 1.1 ratio for both objectives. We assessed scenario-wise conclusions for a given instance configuration using the most adequate aggregation approach, depending on the environment-wise results. For more general conclusions across all instance configurations, we conducted a rank sum analysis of the aggregation.

### 4.3.2 Results

We started our analysis with the help of solution quality over time (SQT) plots, given in Figures 4.2 and 4.3, which respectively depicted instances configured with $\xi = 20\%$ and $\xi = 80\%$. In both figures, a RUE instance was given on the top row, whereas a TSPLIB instance was given on the bottom row. On the left column, instances were configured with $f = 2$, whereas the right column depicted instances configured with $f = 10$. Note that the number of environments had a stronger influence on the final performance of the algorithm, since more environments meant less time for re-optimization. Conversely, the degree of dynamism influenced the solution quality recovery after an environment, since the problem changed more or less drastically depending on this parameter.

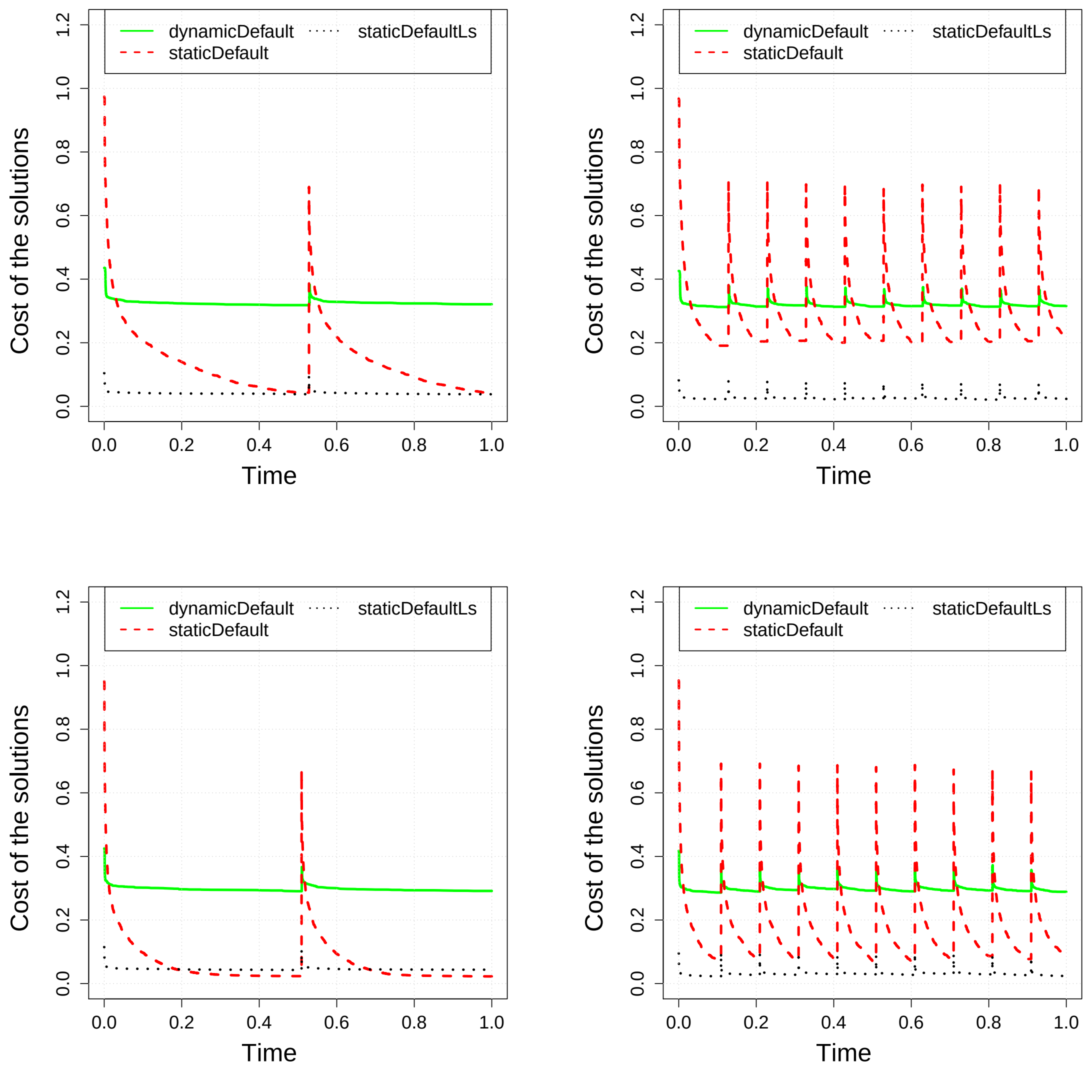


Figure 4.2: SQT plot depicting the anytime performance of P-ACO, on RUE instance size 3500 (top row) and on TSPLIB instance pr2392 (bottom row). Left: scenario $\xi 20 f 2$. Right: scenario $\xi 20 f 10$.

We focused the remainder of the analysis on the most relevant insights from the direct comparison between (i) the two variants that do not use local search, and (ii) between the variants that did not use local search and the one that did.

**Experiments without local search.** The first, contrasting difference between

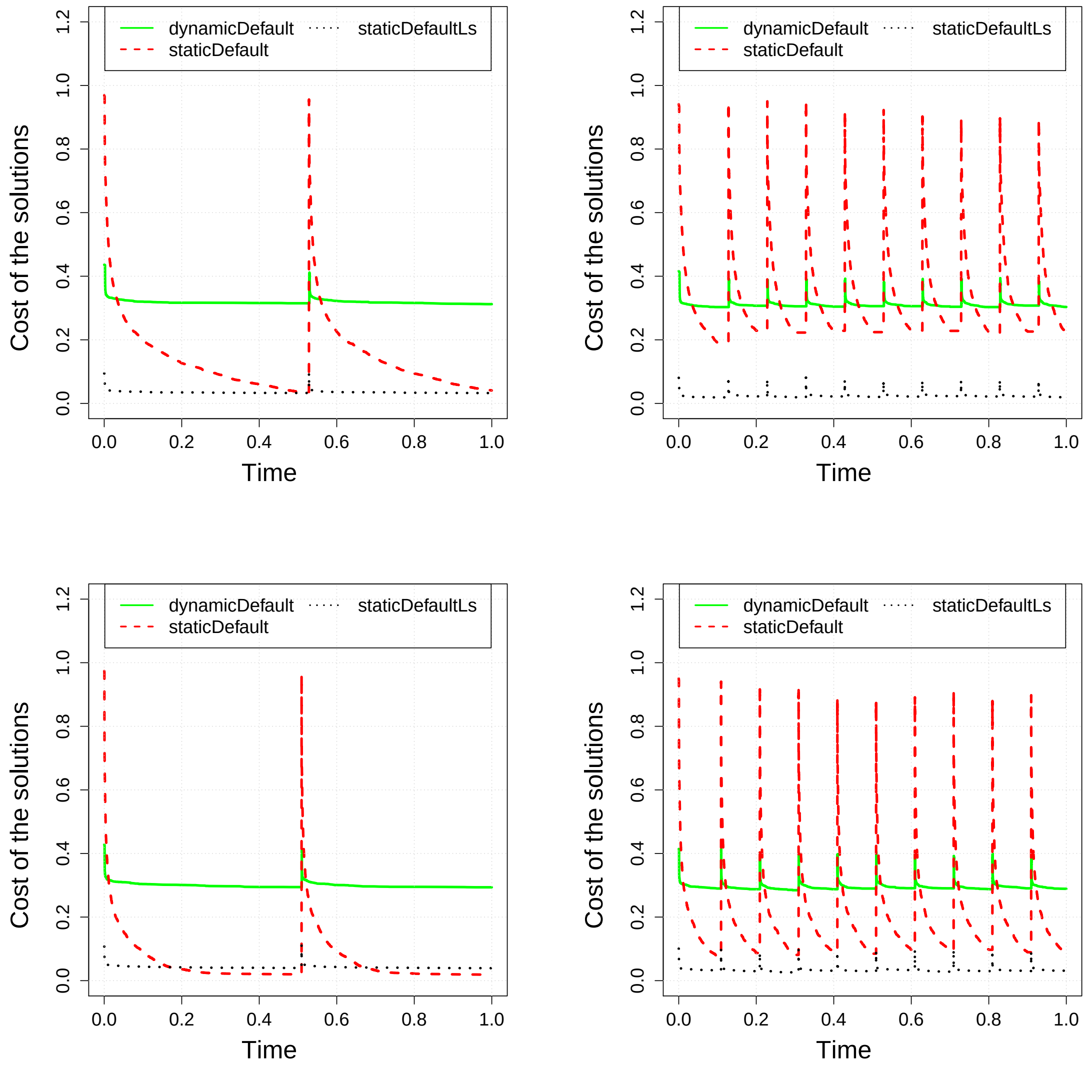


Figure 4.3: SQT plot depicting the anytime performance of P-ACO, on RUE instance size 3500 (top row) and on TSPLIB instance pr2392 (bottom row). Left: scenario $\xi 80 f 2$. Right: scenario $\xi 80 f 10$.

the P-ACO variants that did not use local search, was that the one configured for dynamic optimization was much faster in (re-)optimizing solutions, yet reached a much worse solution quality when compared to the variant configured for static optimization. This was a very interesting result, as it corroborated that algorithms applied to DCOPs need to be engineered with anytime behavior in mind, rather than just being fast or achieving a good final solution. Another interesting observation was that both variants reacted to the randomly produced changes in a very uniform pattern, and indeed the hypervolumes for all environments were very similar. In terms of anytime behavior, the variant configured for static optimization dominated a larger hypervolume, a fact confirmed by the environment-wise hypervolume computation, whatever instance

Table 4.2: Statistical analysis of the different P-ACO variants on RUE and TSPLIB instances aggregate over all instance configurations considered. Each run of an algorithm is ranked according to the average of the environment-wise hypervolumes they dominated. $\Delta R$ gives the difference of the sum of ranks that is statistically significant according to Friedman's non-parametrical test with a confidence level of 95%. The best variant, that is significantly different from the others, is indicated in bold face.

| Instances | *dynamicDefault* | *staticDefault* | ***staticDefaultLs*** | $\Delta R$ |
|---|---|---|---|---|
| RUE + TSPLIB | 51 | 32 | **19** | 3.94 |

configuration considered.

**Experiments with local search.** The overall improvement provided by local search was remarkable for most of the instance configurations considered. Indeed, the variant with local search was at least as fast as the variant configured for dynamic optimization, and reached a final solution with at least the same quality as the variant configured for static optimization. A few factors affected this pattern to some extent. For instance, a larger number of environments further enhanced the benefits of local search, a counterintuitive result given that less runtime was available and local search was known to be computationally costly. Conversely, the benefits for TSPLIB instances were less than for RUE instances, an understandable pattern given that the neighborhood operator we adopted was not particularly effective for TSPLIB. Indeed, when run on TSPLIB instances configured with $f = 2$, P-ACO retrieved better final solutions for each environment when not using local search. Yet, for all but two instance configurations considered, the environment-wise hypervolumes favored the variant that used local search, indicating that it presented a better anytime behavior than the remaining variants.

Table 4.2 shows results that aggregate from all instance configurations considered. Each column depicts the rank sum achieved by each P-ACO variant assessed, and the last column ($\Delta R$) gives the difference in ranks above which the lowest-ranked algorithm can be considered statistically significant better than the remaining algorithms, according to Friedman's non-parametrical test with 95% confidence. Results show a clear separation between variants, and as expected the one that uses local search achieves a lower average hypervolume much more often than the remaining ones.

The fact that the hypervolume indicated an algorithm as having better anytime behavior when it did not reached the best solution quality at the end of an environment (or the run) was a likely possibility. However, as already investigated elsewhere (Radulescu et al., 2013), it was possible to adopt the weighted variant of the hypervolume indicator when one wanted to change the importance distribution of the different regions of the SQT plot. A second important observation was that the clear performance patterns from each algorithm across environments meant that the algorithms that dominated larger hypervolumes could be guaranteed to not have worse anytime performance than algorithms with smaller hypervolumes. In this context, averaging hypervolumes from the different environments was only helpful for the rank sum analysis we conducted next.

Surprisingly, the variant configured for static optimization performed better than the one configured for dynamic optimization. More importantly, we have seen that local search is a crucial component even in the context of dynamic optimization, leading to the best anytime behavior on most of the experiments conducted.

# Chapter 5

# Manual Tuning

When tackling Combinatorial Optimization Problems, the correct setup of its solving algorithm is crucial. Even if the algorithm has a default parameter set, maximizing its performance may involve a proper setting, specifically for each problem tackled.

For many years, the process of finding the best parameter configuration of an algorithm, also known as parameter tuning, has been done manually, according to researches experience or following attempts from more systematically techniques that arose from the field of design of experiments (DOE) (Coy et al., 2001; Adenso-Diaz e Laguna, 2006; Ridge e Kudenko, 2007). In spite of being an attempt to properly configure parameter values, two main issues have arisen from its application. The first one was concerned with the need of deeply knowing the algorithm's search behavior and how its parameters interact. Because some parameters must interact with others, changing the value of one will affect the value applied to the other, which makes this setup adaptation even more complicated. The second relied on the fact that they are time-consuming techniques. Thus, tests need to be run for a small number of instances and approaches. Hence, the set of parameters and search space need also to be small. Furthermore, the instance set used for the tuning process is usually the same used to analyze the performance of the algorithm.

As a consequence, over the last years, the development of tools that were able to automatically configure (automatic tuning) sets of parameters has gained some researches attention. In addition, the application of these tools could also be extended for design space exploration and exploitation, or promoting the switch between algorithms procedures.

Next, we present experiments conducted in order to manually define a suitable parameter configuration of P-ACO algorithm when applied to the TSP with and without the use of local search procedure. These experiments were essential to justify the importance of tuning an algorithm and how this process could be less time-demanding whether an automatic configuration tool had been applied. After that, we depicted the main formulations of automatic tuning presented in the literature applied for COPs. We emphasized the description of the Irace tool, which was the one that we chose to be applied in this research. In the end, we described how to apply the Irace tool for multi-objective tuning. Our objective here is the development of a configuration set for $\mathcal{MM}$AS and P-ACO tool to ACO algorithms when applied to DCOPs aiming at an anytime behavior algorithm performance (Oliveira et al., 2011a,b).

## 5.1 Configurable parameters

Before getting into the tuning process it is important to mention how parameters are configured in this field. Configuration parameters can be primitively defined as numerical or categorical with or without independence and interaction. In the following all these categories are briefly described:

- **Numerical parameters:** a parameter is considered as numerical whether its domain is a real-valued parameter or an integer valued parameter. As an example, considering $\mathcal{MM}$AS, the number of ants $m$ in a colony and the evaporation rate $\rho$ are both classified as numerical parameters;
- **Categorical parameters:** they are represented by discrete values with no ordering relation. For instance, there is the setting of different neighborhood operators available for a local search algorithm, like 2-opt and 3-opt.

It is also important to distinguish the concepts of parameters independence and interaction.

- **Depended or conditional parameters:** when a parameter is used only if specific values for other parameter(s) is (are) selected. As an example, for $\mathcal{MM}$AS nearest neighbor parameters are applied whenever ls is activated.
- **Parameters with interaction:** given two parameters, whenever a change happens in one of them, this action will effect both simultaneously. For example, the intensification and diversification rate used on the multi-colony $\mathcal{MM}$AS which will be described in Chapter 7. Given a random value set to $\rho$ to the first half of ants in the colony $\rho_{fst}$, where $\rho_{fst} = [0, 1]$. The $\rho$ value to the other half is supposed to be set to $\rho_{scd} = 1 - \rho_{fst}$. Parameters that interact cannot be configured independently.

## 5.2 The use of Manual Tuning on P-ACO algorithm

Our first attempt to understand the influence of parameter settings for a particular algorithm applied to a specific problem was the analysis of P-ACO main parameters when applied to the classic TSP. As already mentioned, P-ACO was developed with the possibility of being applied different pheromone update strategies. These strategies were presented and detailed in Chapter 3. Here we chose the *age-based strategy*, which was the first one implemented on P-ACO. It was also the strategy with the simplest behavior to address and the one with the least number of parameters, constraints that were crucial for supporting our studies on parameter configuration. The complete analysis of the experiments is presented in Oliveira et al. (2011b). In this paper, the same study was also presented for the Quadratic Assignment Problem. Here, we presented only the experiments conducted and the main conclusions obtained considering the TS because it is the problem class tackled in this work.

### 5.2.1 Analysis of P-ACO Specific Parameter Settings

In an attempt to support the objectives above mentioned, several experiments were performed on P-ACO with and without the 3-opt local search. The constant parameters used in P-ACO are the same as the ones presented in Guntsch (2004): $m = 10$ ants, $\beta = 2$ and $\alpha = 1$. For $\mathcal{MM}$AS without local search, we set $m = n/4$, in which $n$ is the instance size, $\beta = 2$, $\alpha = 1$, $\rho = 0.02$ for small size instances, $\rho = 0.05$ for medium size instances and $\rho = 0.2$ for large size instances. When local search was applied, the following values were set in MMAS: $m = 25$ and $\rho = 0.2$. The TSP instances used in this research were taken from the TSPLIB: eil101, d198, kroA200, rd400, d657, u724, pcb1173, u1817,d2103, u2319 and are available at `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/`. We grouped the first three instances as small instances, the following three as medium size instances, and the remaining ones as large instances. All results reported for P-ACO, when applied to the TSP, were implemented and executed on top of the ACOTSP software package available at `http://www.aco-metaheuristic.org/aco-code/`. Hence, the P-ACO algorithm here shares the same data structure and speed up techniques as $\mathcal{MM}$AS.

Experiments conducted in this work were run on Intel Xeon 2.4 Ghz quad-core CPUs with 6MB cache and 8 GB of RAM running under Cluster Rocks Linux. The algorithms studied are coded in `C` and compiled with `gcc` version 4.1.2. For the TSP, we set the stopping criteria at 100 seconds for small instances, 1000 seconds for medium instances, and 2000 seconds for large instances.

It is important to note that the plots chosen to be presented in this study were just a sample set to represent our findings. Other plots can be found at `http://anonymous/supp/tobeupdated`.

This experimental analysis was divided into two phases: first, we analyzed the P-ACO specific parameters; second, we proposed an improvement of P-ACO.

The P-ACO age-based algorithm has three specific parameters: $K$, $\tau_{max}$, and $\tau_{min}$. To understand how they influence the algorithm behavior, especially concerning the pheromone update procedure, we tested the algorithm with different values of $K$ and $\tau_{max}$. $\tau_{min}$ is always set to $1/(n-1)$, which is also the initial pheromone value $\tau_0$. Initially, the values we examined were the same as those proposed by Guntsch (2004), that is, $K = 1$, 5, and 25, and $\tau_{max} = 1$, 3, and 10.

However, some additional values were proposed throughout the work, to clarify the observations taken from the results achieved by the initial parameters setting. For this analysis, we considered only the *age-based strategy*, which is the simplest method implemented to manage the solution archive. The impact of using different strategies to manage the solution archive and the algorithm's performance will be presented later.

#### 5.2.1.1 Analysis of K

The plots in Figure 5.1 show that, without the use of local search, P-ACO performed the worst when $K = 1$, and the best when $K = 25$, followed closely by $K = 5$, with negligible differences. Similar behavior was observed across all instance sizes tested. This behavior can be justified by the fact that P-ACO is an algorithm that

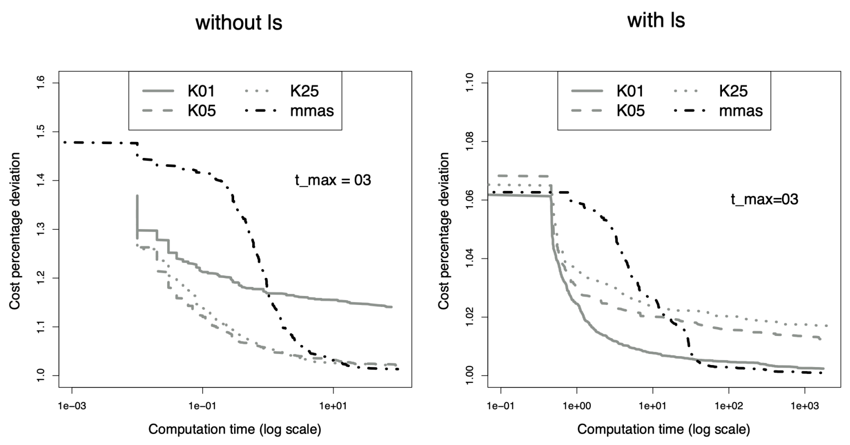


Figure 5.1: Solution quality over time of $\mathcal{MM}$AS and P-ACO using different values of $K$, without local search on instance kroA200 and with local search on instance u1817.

was developed to focus on exploitation. Thus, a larger solution archive enabled the algorithm to better explore the search space.

P-ACO showed the opposite behavior when local search was applied. The best value of $K$ now became one. We could conclude that the search intensification of a local search around one solution was enough and also important to guide the algorithm through high-quality solutions. Note that in Figure 5.1, $\tau_{max}$ is 3. The choice of this parameter value is justified in the following.

#### 5.2.1.2 Analysis of $\tau_{max}$

The ratio between $\tau_{max}$ and $\tau_{min}$ for P-ACO is approximately $\tau_{max} \cdot n$. Thus, the amount of pheromone used to deposit in the pheromone matrix increased linearly according to the instance size, the maximum pheromone trail being $\tau_{max}$. Note that this behavior is similar to the one in $\mathcal{MM}$AS. For more details we refer the reader to Stützle e Hoos (2000).

In Figure 5.2, we had an improvement on the final solution quality on the TSP when $\tau_{max}$ was equal to 3 or 10. On the other hand, no substantial differences could be seen with the use of a local search. The results showed that $\tau_{max} = 3$ was able to improve the algorithm's solution quality for the best values of $K$ in comparison to $\tau_{max} = 1$. No improvement could be seen in the final solution quality for larger values of $\tau_{max}$ and when local search was applied. We also tested $\tau_{max} = 50$.

Overall, the ratio between $\tau_{max}$ and $\tau_{min}$, together with the setting of $K$, influenced how strong the search intensification could be. Given P-ACO and $\mathcal{MM}$AS speedup dependence analysis presented in Oliveira et al. (2011b), we could now visualize the speed advantage of P-ACO with regarding the TSP was much reduced when local search was applied.

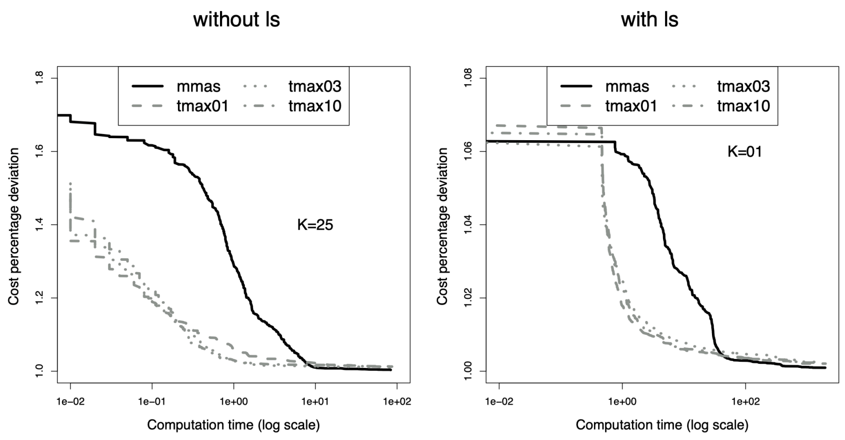


Figure 5.2: Solution quality over time of $\mathcal{MM}$AS and P-ACO using different values of $\tau_{max}$ for the best setting of $K$, without local search on instance kroA200 and with local search on instance u1817.

### 5.2.2 Influence of using Different Strategies to Update the Solution Archive

Another particularity of P-ACO was the way it updated the solution archive. To improve the performance of P-ACO, different strategies to manage the solution archive have been proposed. To compare their influence on the algorithm performance, we tested three strategies from those presented in Guntsch (2004). All strategies tested here were previously explained in Chapter 3. We used, for each strategy, the best values of $K$ and $\tau_{max}$ found in Subsections 5.2.1.1 and 5.2.1.2.

Here $\tau_{max}$ was set to 3, and $K$ was set to 25 when local search was not applied and to 1 when it was applied.

To define the best weight for the *elitist-based strategy*, we tested three values $w_e =$ 0.25, 0.5 and 0.75 for both problems. These weights were the same as those proposed in Guntsch (2004).

In Figure 5.3, we presented the solution development plot of P-ACO with the best parameters among those tested for each population update strategy. For the TSP, the *quality-based strategy* showed the best performance. The *elitist-based strategy* with the right weight (in this case is 0.50 and 0.75), obtained good results too. From all instances tested, P-ACO often outperformed $\mathcal{MM}$AS, independently of the strategy or whether local search was implemented, especially when short computation time was used.

Using the pheromone update strategy as suggested by Guntsch (2004), we managed to obtain similar results. In many cases, P-ACO showed a better performance than $\mathcal{MM}$AS after a short computation time. A concrete comparison to Guntsch (2004) could not be made because of the lack of actual data from the experiments presented.

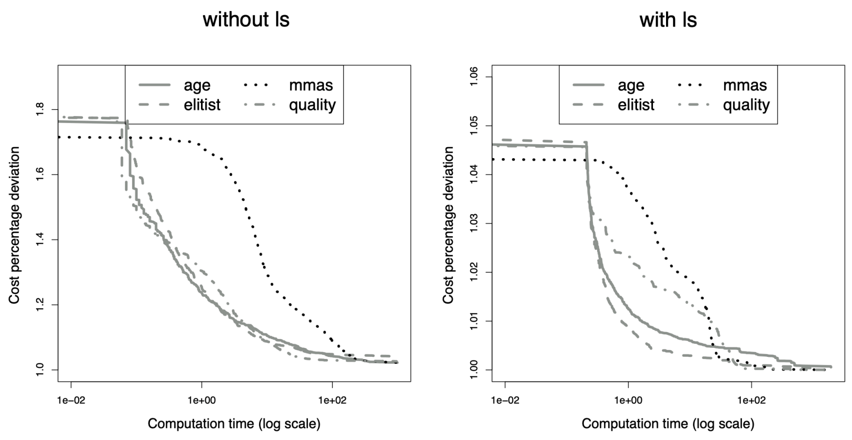


Figure 5.3: Solution quality over time of $\mathcal{MM}$AS and P-ACO considering all the strategies tested, without local search on instance d198 and with local search on instance pcb1173.

#### 5.2.2.1 P-ACO with Restart

P-ACO is an algorithm that presented a strong exploitation capability and, hence, a fast convergence to a right quality solution. On the other hand, its exploration during the search could be insufficient. Following other ACO algorithms such as $\mathcal{MM}$AS we implemented a restart procedure.

It consisted of re-initializing the pheromone matrix values to $\tau_0$ after a certain number of iterations $r$ without improvement. We tested this procedure over a range values of $r$. We tested the pheromone re-initialization after $n$, $25n$, $50n$ and $100n$ iterations when local search was not applied and after 100, 250, 500 and 1000 iterations when it was applied.

Plots in Figure 5.4 show the results achieved using $r = 50n$ when local search is not applied and $r = 500$ when local search is applied. We chose those limits as they were the ones in which the algorithm shows the best overall results.

In Figure 5.4, an improvement in the performance of P-ACO can be seen when we compare the final solution quality achieved by the same algorithm with and without the restart procedure, especially when local search is applied. As shown in the previous Section, the algorithm exploitation capability remains. However, the use of a restart procedure is able to improve P-ACO finally reached solution quality.

#### 5.2.2.2 Statistical Test

In this Section, we compare the finally reached average solution quality between P-ACO and $\mathcal{MM}$AS with and without local search. The percentage deviation is deemed significant if the p-value according to pairwise Wilcoxon signed rank test with Holm correction is less than 0.05. The P-ACO algorithms chosen to be tested are the same as the ones presented in Section 5.2.2.1 as they are the P-ACO variants that achieve best performance. The instances that we used in this section are different from

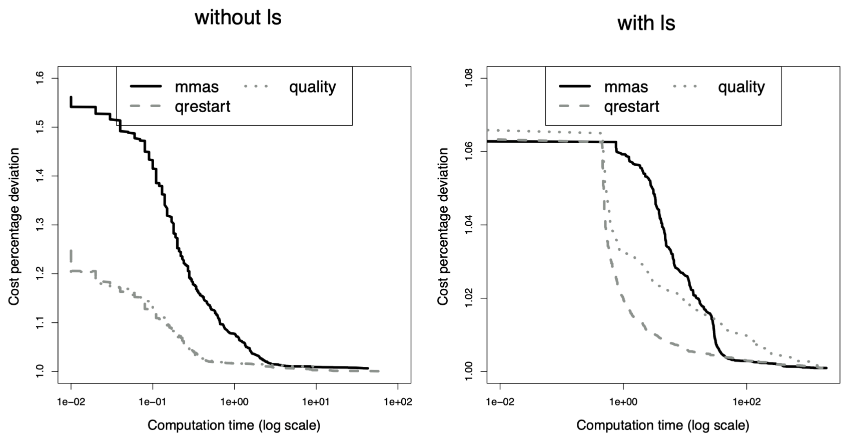


Figure 5.4: Solution quality over time of $\mathcal{MM}$AS and P-ACO considering the quality strategy with the restart procedure, without local search on instance d198 and on instance pcb1173 with local search.

| **Inst.** | **B.known** | **P-ACO** | MMAS | **P-ACO+ls** | MMAS+ls |
|---|---|---|---|---|---|
| kroA100 | 21282 | 0.00 | 0.01 | - | - |
| kroA150 | 26524 | **0.52** | 0.97 | - | - |
| rat195 | 2323 | **0.52** | 0.59 | - | - |
| pcb442 | 50778 | **0.92** | 2.05 | - | - |
| d493 | 35002 | **1.92** | 2.40 | - | - |
| rl1889 | 316536 | - | - | **0.15** | 0.26 |
| u2152 | 64253 | - | - | 0.18 | **0.13** |
| pr2392 | 378032 | - | - | 0.16 | 0.13 |
| pcb3038 | 137694 | - | - | 0.24 | 0.25 |
| fnl4461 | 182566 | - | - | 0.33 | 0.36 |

Table 5.1: Deviation of the average solution quality obtained by P-ACO and $\mathcal{MM}$AS across 30 trials from the best known solution to date. Entries in bold refer to entries with significant difference to the other algorithm.

those used in the previous ones. For the TSP, they were also taken from TSPLIB and are shown in Table 5.1. The P-ACO and $\mathcal{MM}$AS algorithm's configuration remains the same as we presented in Section 5.2.2.1.

The results presented in Table 5.1 show that for the TSP without local search, P-ACO performs better than $\mathcal{MM}$AS except on instance kroA100. Nevertheless, the performance of $\mathcal{MM}$AS and P-ACO are alike when applied to large instances tested with local search.

Overall, these results show that P-ACO with the restart procedure appears to be competitive to $\mathcal{MM}$AS, independently of the local search application.

### 5.2.3 Conclusions

Here, we discussed and analyzed the P-ACO algorithm for the TSP. Extensive experiments were conducted to investigate the behavior of P-ACO when different parameter settings and pheromone update strategies were used. We considered different population update strategies for P-ACO to investigate the effect of using the strategies for solving the TSP. We found that the *quality-based strategy* is the best option for solving the TSP, on the use of local search. Without local search, the *quality-based strategy* appears to provide the best overall tradeoff, while with local search, the *age-based strategy* appears to be a better choice.

The insights that the usage or not of a local search has strong impact on parameters settings for P-ACO applied to the TSP are also new. For example, for the case with local search, a population size of one is clearly the best for P-ACO.

Extensive analysis of the development of solution quality over time shows that in many cases P-ACO outperforms $\mathcal{MM}$AS for short computation times. P-ACO, however, shows an early stagnation behavior compared to $\mathcal{MM}$AS. To overcome this, we introduced a restart mechanism, and this improved significantly the overall performance of P-ACO.

As the main result, we can conclude that, with the restart procedure and the right configuration, P-ACO is competitive to the state-of-the-art ACO algorithms with the advantage of finding good solution quality in a shorter computation time. However, we believe that using an automatic configuration tool could improve the configuration proposed here, at the same time bringing more and deeper insights about the algorithm search behavior.

# Chapter 6

# Automatic Tuning

As presented in Chapter 5, tuning an algorithm, besides being time-consuming, demands an intuitive range selection for each parameter tested. It can lead researchers to incomplete or even erroneous conclusions about the algorithm tested performance if they do not have deep knowledge about the algorithm procedures and how they interact, which will change according to the problem tackled. As an example, when we moved from classic TSP to the TSP with dynamic demands, P-ACO would need to handle its exploitative behavior especially right after the dynamic changes. On the other hand, $\mathcal{MM}$AS would need to speed up ants exploration and ants memory would need to be adapted given the shorter amount of run time and the dynamic changes over the instances tested.

Thus, in this Chapter, we focused our attention on the application and analysis of the Irace automatic tuning tool for P-ACO and $\mathcal{MM}$AS. In order to do that, we firstly present an overview of the automatic tuning process with emphasis on the Irace tool which was the tuning tool that we have chosen to be applied here. After that, the Irace tool is applied on both algorithms and their performance is compared. Both P-ACO and $\mathcal{MM}$AS were tuned separately with and without using the local search procedure. Our objective here was to identify whether different configurations were also needed when isolating the use of the local search procedure.

## 6.1 Offline automatic tuning tool

The literature has presented two ways of automatically tuning an algorithm. They were:

**Offline automatic tuning:** Here the benchmark set used is divided into a *training set* and a *testing set*. The tuning process is applied on a training benchmark set, and the algorithm performance from the parameter settings is evaluated according to a different benchmark set called the testing set (Pellegrini et al., 2010).

**Online automatic tuning:** In this approach, the adaptation of the parameters happens while solving the instance benchmark. Thus, the configuration might be changed during the solution process to adapt to the instance being solved and possibly to the state of the search. However, even *online tuning* has *offline* *tuned parameters*, like the ones that are related to the tuning design choices, as presented in López-Ibánez e Stützle (2014); López-Ibáñez et al. (2016).

In this research we focus our attention on the *offline tuning*. Once *online tuning* needs offline tuned parameters, we believed that *offline tuning* would be the preliminary way off tuning a DCOP, which latter, could be followed by the application of the *offline tuning* procedure.

Let us assume that we have a parametrized algorithm with $n^{par}$ parameters, $X_p$, $p = 1, .., n^{par}$, and each of them can assume different values. A configuration of an algorithm $\theta = \{x_1, ..., x^{par}\}$ is a unique assignment of values to these parameters, and $\Theta$ denotes the possibly infinite set of all configurations of the algorithm. When applied to an instance, each configuration of $\theta$ has a cost value assigned, that is $C(\theta, I)$. The cost can be represented as the best solution quality obtained from the objective function within a computation time. Thus, the cost measure assigns a cost value to one run of the algorithm with a specific configuration on a specific instance. The goal of the automatic tuning is to find the best configuration $\theta^{\star}$that minimizes $F(\theta)$. Given the fact that the number of configuration sets $\Theta$ are infinite, they are generated by sampling.

## 6.1.1 Racing approaches

The evaluation of configurations is typically the most computationally demanding part of an automatic configuration method. To handle this issue, racing methods (Birattari et al., 2002) can be a good strategy for selecting a configuration among a number of candidates using sequencial statistical testing. The initial candidate configurations for a race may be randomly selected, based on problem specific knowledge, or based on DOE techniques (López-Ibáñez et al., 2016)

The racing approach originated from the machine learning literature (Birattari et al., 2002). This approach was later adapted to make it suitable for the configuration task. At each step, a given finite set of candidate configurations are evaluated in parallel and candidate configurations are discarded as soon as statistical evidences prove that they are poor. The elimination of poor candidates allows the tuning to be concentrated on candidates that are promising.

### 6.1.1.1 F-Racing approaches

The F-Race is a racing algorithm based on the non-parametric Friedman's two-way analysis of variance by ranks (Friedman test) proposed by Birattari (2009); Birattari et al. (2010).

The racing algorithm evaluates a given finite set of candidate configurations step by step. At each step, an instance is tested over all candidate configurations. These configurations are then ranked and evaluated in parallel and the poor candidate configurations are eliminated according to Friedman's tests. The elimination of poor candidates allows to focus on the most promising ones. The number of steps, that is, the number of instances tests are adaptively adjusted based on statistical evidence. The above racing procedure stops either when only one candidate configuration lasts, a given maximum number of instances have been sampled, or when the predefined computational budget has been achieved.

According to his approach, the Friedman test assumes that costs generated by each configuration tested are mutually independent.

#### 6.1.1.2 F/I-Racing approach

Iterated racing as presented in Birattari et al. (2010); López-Ibáñez et al. (2016) is a method for automatic configuration that consists in three steps:

1. **Sampling**: In the first iteration, the initial set of candidate configurations is generated by uniformly sampling the parameter space *Chi*. Non-conditional parameters are sampled first, those parameters that are conditional to them are sampled next, if the condition is satisfied. At each iteration, a number of surviving candidate configurations from the previous iteration bias the sampling of new candidate configurations. In spite of having different ways to generate the first sampling set of parameters, it is suggested to be generated randomly.

2. **Selection**: In each iteration the best configurations from the newly sampled ones are selected by means of racing. As already mentioned, racing was first proposed for machine learning proposals and it was later adapted for configuration of optimization algorithms. Given the set of candidate configurations the scheme of racing procedure applied is presented in Figure 6.1.

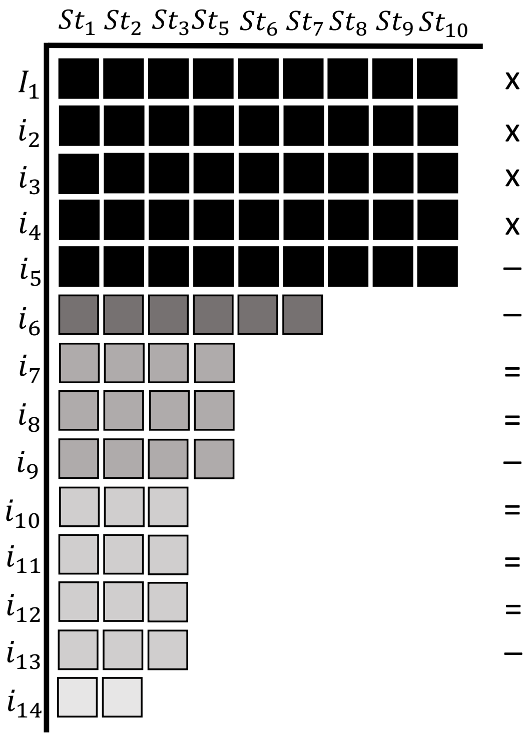


Figure 6.1: Racing for automatic algorithm configuration.

The race starts with a finite set of candidate solutions. The Figure shows ten sets $St_1, ..., St_{10}$, at each step of the race an instance $I$ is run over all these configurations. When the race starts each configuration is evaluated on the first instance $I_0$ by means of the cost value $C$. Configurations are iteratively evaluated on subsequence instances until a number of instances have been seen. Then, the candidate solutions are ranked and a statistical test is applied. Candidate configurations that perform statistically worse than at least another one are removed, and the race continues with the remaining configurations. After the first statistical test, the next ones are performed at every $n_{step}$ step. This process continues until achieving a stop criteria that could be: (i) reaching a maximum number of surviving configurations; (ii) whether all instances had been analyzed or (iii) reaching a pre-defined computation budget (time spent or number of iterations). As presented in Figure 6.1, the first statistical test is executed at step 5, that is, after running the configurations sets on instance $I_5$. By default, after the first test, the next ones are performed at each instance. At the end of a race, surviving configurations have a rank assigned $rk_z$, configurations with the lowest rank are selected as the set of elite configurations $\Theta^{elite}$. For generating a new configuration with a probability higher, ranked configurations have a higher probability of being selected as parents of new candidate configurations.

3. **Adaptation**: In the next iteration, before the race starts, a number of new candidate configurations are updating the sampling distribution in order to bias the sampling towards the best configurations. New configurations according to a particular probability of distribution where *numerical parameters* the sampling distribution follows a truncated normal distribution. In this case, the update of distributions consists in modifying their mean and standard deviation. On the other hand, *categorical parameters* are handled by discrete distribution and values are updated according to discrete probabilities. These updates allow the new samplings to be concentrate over time only on promising configurations sets.

These three steps are repeated until a termination condition is not met.

## 6.2 Automatic configuration for Anytime behavior problems

As previously discussed, parameter settings are critical for the performance of metaheuristic-based algorithms such as ACO. Nonetheless, few are the works that conduct rigorous analysis of the parameter configuration best suited for dynamic optimization algorithms. More importantly, transferring parameter settings configured for one setup to be used in another setup is an approach prone to major drawbacks, as the literature on parameter configuration has repeatedly demonstrated (Bezerra, 2016). Specifically for the TSP with dynamic demands, the works presented in the literature have been done in this way (Guntsch e Middendorf, 2001; Guntsch, 2004; Mavrovouniotis e Yang, 2010, 2013b,a, 2014b). Here we proposed an automatic configuration tool that, besides giving an specific complete configuration set for the problem tackled, also enabled the use of P-ACO and $\mathcal{MM}$AS as an anytime behavior algorithm.

Typically, the objectives of a bi-objective problem are as conflicting as those for the TSDDs in which we would like to minimize the run time and the problem objective function. Thus, the solutions lie in a bi-dimensional space.

### 6.2.1 Experiments Setup

Next, we detail the experimental setup we adopted, individually detailing benchmark, performance evaluation, configuration, and testing setups. As recommended, the benchmark and scenarios used for the Irace execution were different from the ones used to test the algorithms' performance after tuning. The selected TSPLIB instances were u724, rl1304, pcb1173, u2319, and pcb3038. The instance name gives the number of cities in the instance. The second set is composed of random uniform Euclidean (RUE) instances, with sizes depending on whether the local search is used or not. Specifically, we use 2 subsets of 5 instances with sizes ranging from 2500 to 3000 when local search is not applied, and 3 subsets of 5 instances with sizes ranging from 3000 to 4000 when local search is applied all of them different from the ones that were used to compare $\mathcal{MM}$AS and P-ACO performance. Dynamic environments were generated using the method described in Mavrovouniotis e Yang (2011b) and also in Chapter 2. In our experiments, here $f$ is set to 4 and 6, respectively. The degree of change is set to $\xi \in \{30\%, 60\%, 90\%\}$, and so the combinations of $\xi$ and $f$ also comprised 6 different scenarios.

### 6.2.2 Assessing parameter settings

Parameter settings used in these experiments are given in Table 6.1. Default settings are given in the top rows, whereas configured settings are given in the bottom ones. In addition, the suffix *LS* is added to settings that are used in experiments in which local search is adopted. Concerning default settings, a few observations stand out. First, despite their different structural characteristics, P-ACO and $\mathcal{MM}$AS have typically been run with the same parameter settings. Second, settings are very similar whether local search is used or not. Finally, the most noticeable difference between settings from the static and the dynamic optimization literature is the value of $\beta$, in an attempt to provide algorithms with stronger convergence pressure and thus speed-up solution re-optimization.

By contrast, the configured settings selected by Irace given in Table 6.1 (bottom) are different for each algorithm, as well as between runs that use local search and runs that do not. Concerning $\mathcal{MM}$AS, for instance, we notice a larger value of $\rho$ that makes the search more explorative. Effectively, this setting helps the algorithm escape from the search space region to where it had converged before an environment change, but had not yet been considered by manual configuration. The value of $\beta$ changes as a function of local search. When local search is not adopted (*mmasTuned*), $\beta$ is also increased to allow the algorithm faster exploitation due to the limited amount of runtime available. Conversely, the strong exploitation nature of the local search component induced a decrement of $\beta$ in *mmasTunedLS*.

Regarding P-ACO, parameters configured by Irace for the experiments without local search (*pacoTuned*) greater resemble the settings adopted in the static optimization literature (*pacoDefault*) than the ones used in the dynamic optimization liter-

Table 6.1: Default (top) and configured (bottom) parameter settings used for $\mathcal{MM}$AS (settings with the *mmas* prefix) and P-ACO (settings with the *paco* prefix). The *LS* suffix is appended to settings used on experiments where local search is adopted. We remark that settings for *pacoDynamicLS* are not given since no study has yet investigated P-ACO coupled with local search for dynamic optimization.

| | $\mathcal{MM}$AS + P-ACO | | | $\mathcal{MM}$AS | P-ACO | |
|---|---|---|---|---|---|---|
| Settings | $m$ | $\alpha$ | $\beta$ | $\rho$ | $\tau_{max}$ | $K$ |
| mmasDefault | $n/4$ | 1 | 2 | 0.2 | – | – |
| mmasDefaultLS | 25 | 1 | 2 | 0.2 | – | – |
| mmasDynamic | 50 | 1 | 5 | 0.2 | – | – |
| mmasDynamicLS | 50 | 1 | 5 | 0.2 | – | – |
| pacoDefault | $n/4$ | 1 | 2 | – | 3 | 25 |
| pacoDefaultLS | 25 | 1 | 2 | – | 3 | 1 |
| pacoDynamic | 50 | 1 | 5 | – | 3 | 3 |
| mmasTuned | 96 | 1 | 5 | 0.6 | – | – |
| mmasTunedLS | 9 | 1 | 1 | 0.4 | – | – |
| pacoTuned | 79 | 1 | 3 | – | 3 | 25 |
| pacoTunedLS | 5 | 2 | 2 | – | 3 | 1 |

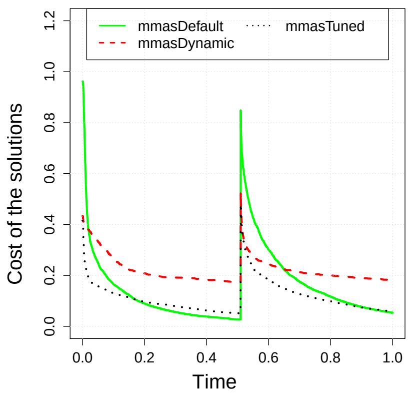
ature (*pacoDynamic*). Interestingly, this resemblance between static and configured settings also holds in the presence of local search (*pacoDefaultLS* and *pacoTunedLS*).

To empirically assess the impact of the different parameter settings presented in Table 6.1, we discuss below the most important insights observed with the help of SQT plots.

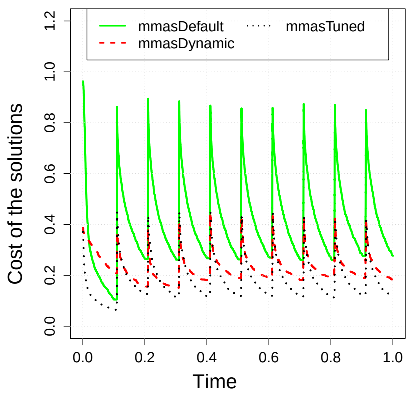
- **Dynamic characteristics of the scenarios**

  Figure 6.2 shows the anytime performance of $\mathcal{MM}$AS (top) and P-ACO (bottom) on a TSPLIB instance, run without local search using the three different parameter settings on two different experimental scenarios. In common, both scenarios present a degree of dynamism $\xi = 40\%$, but differ as to the frequency of change $f \in \{2, 10\}$. The overall pattern depicted in these plots shows how parameter settings can be affected by the dynamic characteristics of the scenarios. Notice, for instance, the anytime performance of $\mathcal{MM}$AS settings on the topmost plots. On the left (Figure 6.2a), the default settings of $\mathcal{MM}$AS (*mmasDefault*) improve over the dynamic settings (*mmasDynamic*), but the opposite happens on Figure 6.2b. In summary, an increment in the frequency of change is enough to alter the relative performance of the settings. The configured settings (*mmasTuned*) represent a compromise solution that shows robust performance across different scenarios.

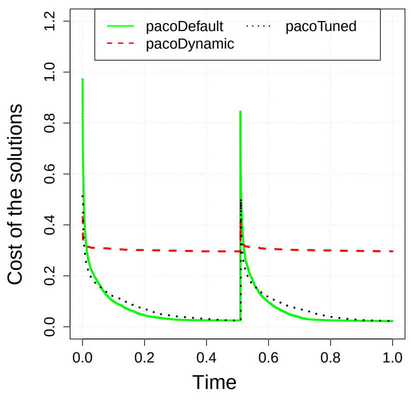
  Regarding P-ACO, the improvements obtained with the help of automatic configuration are significant for most scenarios. Yet, for specific scenarios such as the one depicted in Figure 6.2c, P-ACO with default and configured configuration settings (*pacoDefault* and *pacoDynamic*, respectively) present similar performance. Moreover, we remark that the worst performance observed in Figures 6.2c and 6.2d concern P-ACO run with dynamic settings (*pacoDynamic*), which reinforces the importance of automatic parameter configuration methodologies.



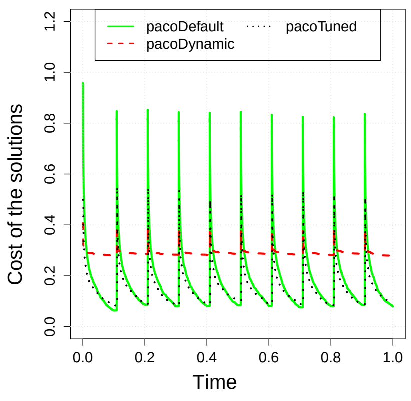
(a) $\xi$=40, $f$=2



(b) $\xi$=40, $f$=10



(c) $\xi$=40, $f$=2



(d) $\xi$=40, $f$=10

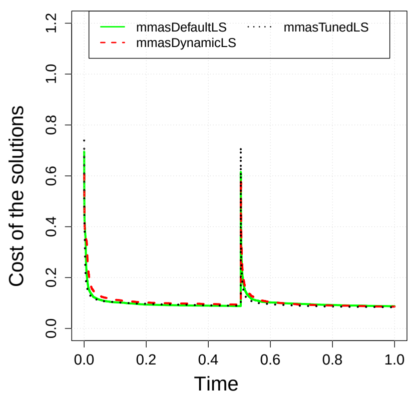
Figure 6.2: SQT plot depicting the anytime performance of $\mathcal{MM}$AS (top) and P-ACO (bottom) on TSPLIB instance pr2392, without local search, run with different parameter settings on different dynamic scenarios.

- **Interactions with local search**

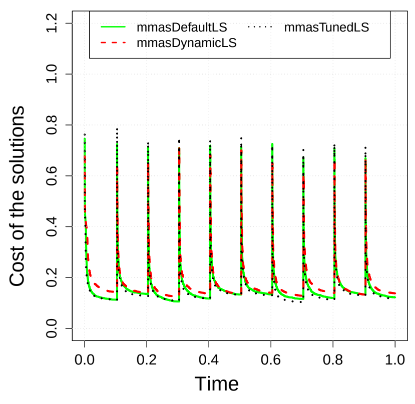
  Figure 6.3 shows the anytime performance of $\mathcal{MM}$AS and P-ACO when run with local search on the same instance and scenarios from the previous analysis. Although the conclusions about the effects of the dynamic characteristics of the scenarios still hold, the use of local search significantly changes the improvement rate provided by automatic configuration for $\mathcal{MM}$AS. This is easily explained by the high-quality solutions obtained when local search is adopted, which makes any further improvement much more difficult to be obtained. The performance of P-ACO, on the other hand, is significantly improved by the configured settings. Furthermore, the benefits of automatic configuration are consistent through both scenarios.

- **Structural characteristics of the benchmark instances**

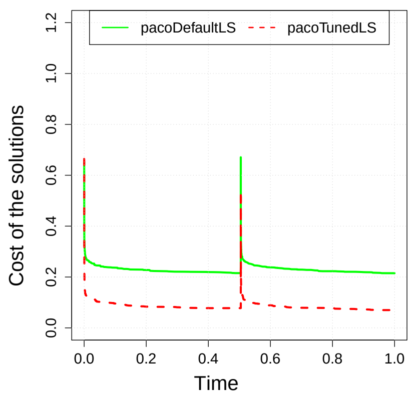
  The differences in benchmark instance characteristics are reflected in the perfor-



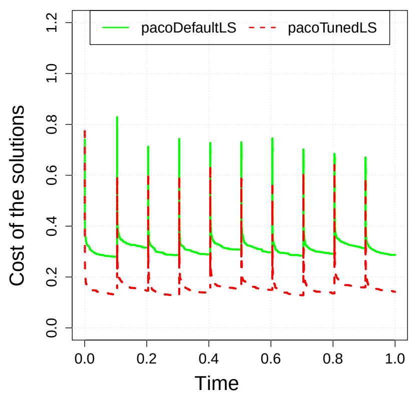
(a) $\xi$=40, $f$=2



(b) $\xi$=40, $f$=10



(c) $\xi$=40, $f$=2

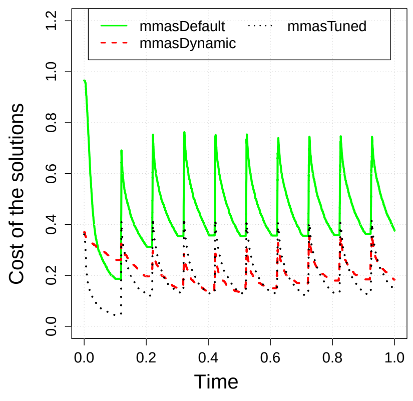


(d) $\xi$=40, $f$=10

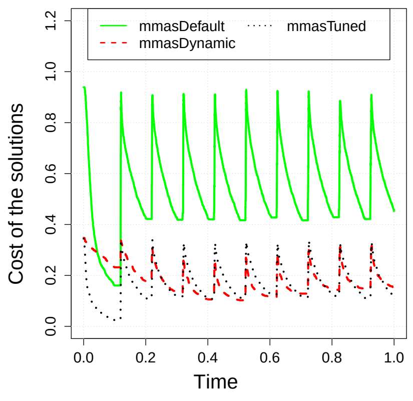
Figure 6.3: SQT plot depicting the anytime performance of $\mathcal{MM}$AS (top) and P-ACO (bottom) on TSPLIB instance pr2392, with local search, run with different parameter settings on different dynamic scenarios.

mance displayed by different parameter settings, at least in the case of $\mathcal{MM}$AS. Figure 6.4 helps explain our observation, showing the performance improvements obtained from automatic configuration when running $\mathcal{MM}$AS on scenarios with $f = 10$ and $\xi = 20\%$ (left) or $\xi = 80\%$ (right). Specifically, results from RUE instances (top) show a much larger improvement provided by configuration, even if the improvement seen on the results from TSPLIB instances (bottom) is also significant.

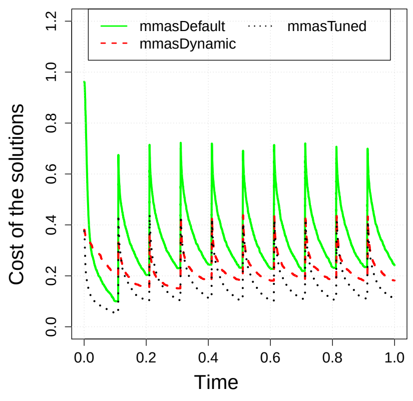
A possible explanation is the importance of cities (customers to be attended) in the different benchmark sets. In more detail, RUE instances are expected to present cities of similar relevance to solution quality, whereas the relevance of TSPLIB instance cities may differ considerably. These results indicate that, though the configured settings are robust w.r.t. benchmark sets, an *a priori* knowledge of these features would likely benefit the configuration process.



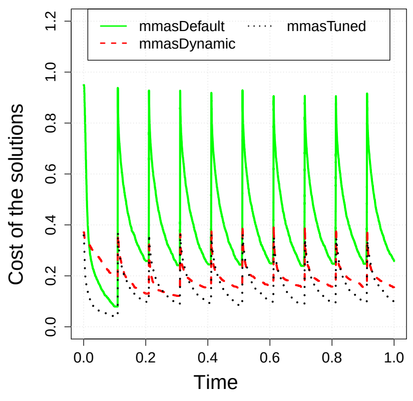
(a) $\xi$=20, $f$=10



(b) $\xi$=80, $f$=10



(c) $\xi$=20, $f$=10

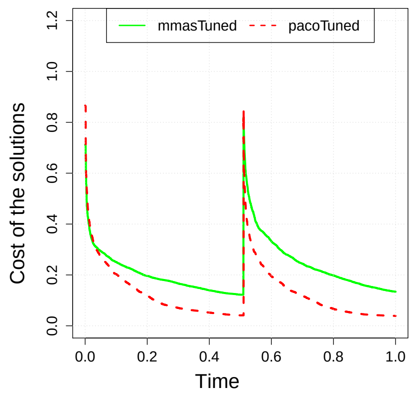


(d) $\xi$=80, $f$=10

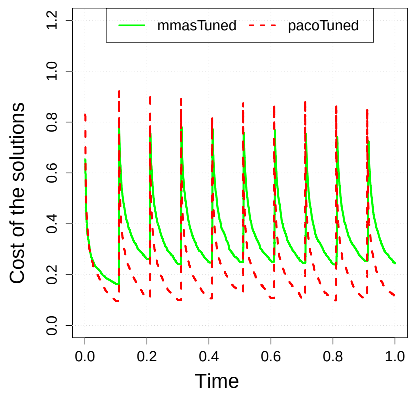
Figure 6.4: SQT plots depicting the anytime performance of $\mathcal{MM}$AS on RUE instance 3002 (top) and on TSPLIB instance pr2392 (bottom), without local search, run with different parameter settings on different dynamic scenarios.

### 6.2.3 Performance comparison

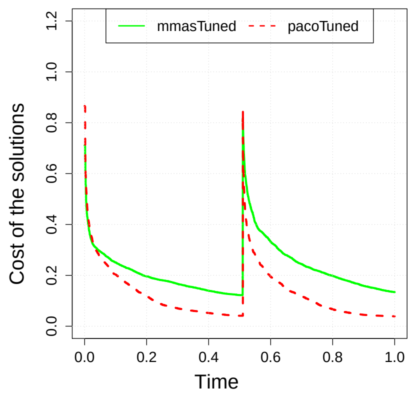
We next directly compare $\mathcal{MM}$AS and P-ACO, once again isolating the effects of local search. For these experiments, however, we consider only the configured parameter settings, as the results from the previous section confirmed that the automatic configuration methodology leads to improvements in the anytime performance of the algorithms. Overall, the insights we observed are considerably different for the experiments with and without local search, corroborating our claim that experimental analyses should isolate this factor. Figure 6.5 illustrates the most important insights we observed from each set of experiments, which we discuss below.



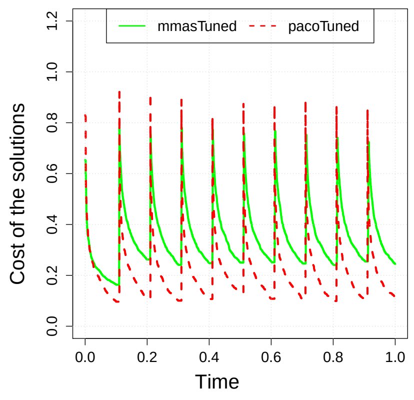
(a) $\xi$=40, $f$=2



(b) $\xi$=40, $f$=10



(c) $\xi$=40, $f$=2



(d) $\xi$=40, $f$=10

Figure 6.5: SQT plots depicting the anytime performance of P-ACO and $\mathcal{MM}$AS configured by irace, on TSPLIB instance pr2392 with (top) and without (bottom) local search run on different dynamic scenarios.

- **Experiments without local search**

  When local search is not adopted (top-most plots), the difference in performance between $\mathcal{MM}$AS and P-ACO across different environments is significant. More precisely, $\mathcal{MM}$AS demonstrates its best performance before the first environment change, whereas P-ACO displays a constantly good performance throughout the run. In addition, the effect for $\mathcal{MM}$AS is much weaker for $f = 2$ (left) scenarios than when $f = 10$ (right). Altogether, these observations can be justified by the characteristics of the ACO algorithms being compared. $\mathcal{MM}$AS is an algorithm originally designed for static, final-quality optimization, and thus it requires a given minimum runtime to produce good results. Moreover, the best applications of $\mathcal{MM}$AS rely on local search to increase its convergence pressure, an aid we specifically forbid in this set of experiments. Concerning P-ACO, its pheromone transfer mechanism and the fast update procedure appear to be the keys for its good performance. In addition, P-ACO studies have only started considering local search recently, and hence, the original algorithm was designed to be effective without this extra source of pressure.

- **Experiments with local search**

  Results change considerably when algorithms are allowed to use local search (bottom-most plots). In particular, both algorithms benefit from the additional convergence pressure, but $\mathcal{MM}$AS is now able to outperform P-ACO by a significant margin across all instance types, sizes, and scenarios. More importantly, $\mathcal{MM}$AS is no longer affected by the runtime available (nor, consequently, by the dynamic characteristics of the scenarios). Analogously, the performance of P-ACO is the same across all environments, whatever the instance type, size, or scenario considered. A deeper analysis revealed that the advantage of the faster pheromone update designed for P-ACO is much reduced, since relatively the local search procedure uses a significant amount of time. On the other hand, $\mathcal{MM}$AS presents speed-ups specifically conceived to improve the efficiency of its coupling with local search procedures, e.g., avoiding updating the pheromone matrix as a whole. Altogether, these changes in the relative efficiency of the algorithms explain the change in their anytime performance, with $\mathcal{MM}$AS clearly becoming the best option for the DTSP when local search is allowed.

### 6.2.4 Statistical analysis

To support the insights discussed in this section, Table 6.2 provides a rank sum analysis where we compare all variants of the algorithms, isolating the effect of local search. In more detail, we conduct three levels of aggregation. At the bottom level, we evaluate an algorithm w.r.t. to the average hypervolume it produces over all environments in a given run. At the mid-level, we evaluate an algorithm w.r.t. its average performance over the 20 runs on a given block, i.e., a given instance from a given experimental scenario. At the top level, for each block algorithms are ranked in ascending order according to their performance, and the sum of the ranks obtained by an algorithm on all blocks depicts its overall performance. These rank sums are used to assess statistical significance with 99% confidence level using Friedman's non-parametric test and associated post-hoc test (Conover e Conover, 1980). The critical difference in ranks indicated by the test is provided as $\Delta R$. Algorithms that present a rank sum difference w.r.t. to the best ranked smaller than $\Delta R$ are highlighted in boldface, indicating that no statistical significant difference was observed between the performance of the given algorithm and the best ranked one.

As discussed from the plots, when local search is not adopted, the performance of P-ACO is very similar when both default and configured settings are adopted. Ironically, the worst performance among all algorithms is observed for the settings typically adopted for P-ACO in the dynamic optimization literature. Regarding $\mathcal{MM}$AS, the configured settings improve over the manually-configured ones, but not enough to match the performance of P-ACO. Conversely, conclusions change completely when local search is adopted. For this setup, $\mathcal{MM}$AS is able to outperform P-ACO no matter the configuration adopted. More importantly, the configured version of

Table 6.2: Statistical analysis of $\mathcal{MM}$AS and P-ACO using different parameter configurations, without and with local search, aggregated over all instances and scenarios considered. Rank sum differences w.r.t. the best ranked algorithm are given in parenthesis.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Without LS** | *Mean* ($\Delta R = 36.77$) | ***pacoTuned*** | ***pacoDefault*** **(28.5)** | *mmasTuned* (142) | *mmasDynamic* (257) | *mmasDefault* (313.5) | *pacoDynamic* (405) |
| **With LS** | *Mean* ($\Delta R = 9.3$) | ***mmasTunedLS*** | *mmasDefaultLS* (110) | *mmasDynamicLS* (190) | *pacoTunedLS* (304) | *pacoDefaultLS* (406) | |

$\mathcal{MM}$AS statistically significantly outperforms both the settings adopted in static and dynamic optimization. Regarding P-ACO, configured settings improve over the settings adopted in static optimization, but not enough to match the performance of $\mathcal{MM}$AS.

### 6.2.5 Conclusions

The experiments comparing the top-performing ACO algorithms from the context of static and dynamic optimization confirmed that the most important factors in such an analysis are the appropriate configuration of parameters and the use of local search. In real-world scenarios, parameter configuration and local search procedures need to be assessed as to their practicality, since it may not always be the case that one is able to run the number of experiments required by offline configuration, or the expected time available for each environment can be too small for local search to be of any benefit. Nonetheless, it is easy to conceive a database of configured parameter settings for different problem benchmarks, scenarios, and stopping criteria, greatly reducing the overhead of offline configuration in practice. In the next section, we investigate the benefits of approaches proposed in the dynamic optimization literature to improve ACO performance.

# Chapter 7

# Improving $\mathcal{MM}$AS for the TSPDD

The successful applications of ACO to static COPs stirred the interest of researchers that investigate dynamic COPs (DCOPs) (Rohlfshagen e Yao, 2008; Yang et al., 2013; Mavrovouniotis et al., 2020, 2017; Mavrovouniotis e Yang, 2018). In contrast to the highly advantageous use of a pheromone memory in static COPs, a dynamic scenario makes the long-term knowledge of ACO a mixed blessing. On one hand, changes to the problem data are expected to only affect a portion of its original definition, and so ACO algorithms could possibly reuse information learned before the changes to speed up the re-optimization cycle. On the other hand, if an ACO algorithm has already converged to a specific region of the solution space that after a change is no longer interesting or even feasible, the algorithm search might be delayed by the need to first forget its previous knowledge.

In this Chapter we conducted a computational study to investigate the role of pheromone transfer with and without local search in the performance of $\mathcal{MM}$AS. We maintained our focus on conducting our computational study on the traveling salesman problem (TSP) with dynamic demands (Guntsch e Middendorf, 2001; Mavrovouniotis e Yang, 2011a; Oliveira et al., 2019).

Our investigation revisited some of the proposals to extend $\mathcal{MM}$AS to DCOPs (Mavrovouniotis e Yang, 2013b, 2011a, 2014a; Mavrovouniotis et al., 2015; Mavrovouniotis et al., 2017; Mavrovouniotis e Yang, 2013a), assessing under our setup the improvements to $\mathcal{MM}$AS performance provided by those pheromone transfer mechanisms.

More importantly, we used $\mathcal{MM}$AS as ACO test benchmark to understand if those components contributed to the performance of effective ACO algorithms in general. Different adaptations of ACO algorithms and procedures for DCOPs have been proposed in the literature, but no study targeting the potential interactions between adaptation proposals has yet been conducted. In this work, we conducted such an investigation specifically targeting the two aforementioned sets of proposals, namely (i) local search, and (ii) parameter settings on different pheromone transfer procedures.

We initially created a set of variants of the selected baseline ACO algorithm, differing by the addition of a single pheromone transfer proposal. Effectively, the assessment of a variant was actually an assessment of how the algorithmic component that characterizes the given variant contributed to the performance of high-performing ACO algorithms in the context of DCOPs. In addition, to understand the importance of the other two experimental factors (local search and parameter settings), each experiment we conducted were performed with and without local search, and with different parameter settings that were obtained from automatic configuration, see Chapter 6.

## 7.1 Evaluation

As previously discussed, algorithms applied to DCOPs have been traditionally compared using different performance metrics. In common, both final-quality based and behavior-based metrics have been indiscriminately assessed concerning number of iterations, function evaluations, or solutions generated (Weicker, 2002; Rand e Riolo, 2005; Guntsch e Middendorf, 2002, 2001; Mavrovouniotis e Yang, 2011b; Sarasola e Alba, 2013; Alba e Sarasola, 2010a; Ben-Romdhane et al., 2013; Nguyen et al., 2012b). The rationale behind these resource consumption metrics was understanding by how much algorithms were able to improve solutions within a given iteration, of after a fixed number of function evaluations / solutions generated. Although they shared the benefit of being hardware-independent (as long as no maximum runtime was established), they also mask the computational overhead of the algorithms, an effect that is particularly undesirable in the context of dynamic optimization. Here, we evaluated algorithms using the hypervolume metric detailed in Chapter 7 considering as resource consumption metric the runtime of the algorithms. Concretely, as same as on previous experiments, algorithms were allowed a maximum runtime of 2 000 seconds when they used local search, and 1 000 seconds otherwise. By doing so, we put on evidence the most important characteristic algorithms designed tackling DCOPs should present, namely to be efficient as to the runtime they required for reoptimizing solutions.

## 7.2 Configuration

As detailed in Chapter 6, automatic algorithm configurators such as Irace should be used within a carefully designed configuration setup. To meet this need, we formally defined a configuration space given in Table 7.1 based on the ACO literature (Dorigo e Stützle, 2004; Guntsch e Middendorf, 2001; Guntsch, 2004; Guntsch e Middendorf, 2002; Mavrovouniotis e Yang, 2011b), delimiting parameters and domains for each algorithm we configured in this investigation.

Concerning the separation between configuration and testing benchmark sets, we created an alternative benchmark set for configuration, comprising thirteen instances ranging from 100 to 3000 cities taken from the TSPLIB instance benchmark (Reinelt, 2008)[1], and 15 RUE instances, ranging from 2 000 to 4 000 cities (http://dimacs.rutgers.edu/archive/Challenges/TSP/, 2018). Additionally, to prevent floor effects that could reduce the effectiveness of the automatic configuration, when configuring an algorithm allowed to use local search we only adopted instances with more than 600 cities. Concerning the configuration budget, Irace is

[1]The selected TSPLIB instances are rd100, kroA150, kroB200, gr202, pr226, pr439, gr666, u724, vm1084, rl1304, vm1748, u2319 and pcb3038.

Table 7.1: Parameters space used as input for Irace when configuring $\mathcal{MM}$AS and P-ACO. For brevity, we use the $(x_i, x_n)$ notation to represent a discrete interval between $x_i$ and $x_n$, where all integer values are considered by Irace.

| Algorithm | $m$ | $\rho$ | $\alpha$ | $\beta$ | $q_0$ | $\tau_{max}$ | $K$ |
|---|---|---|---|---|---|---|---|
| $\mathcal{MM}$AS | $(5, 100)$ | $[0.1, 1]$ | $[0, 5]$ | $[1, 10]$ | – | – | – |
| P-ACO | $(5, 100)$ | – | $[0, 5]$ | $[1, 10]$ | – | $[1, 10]$ | $[1, 25]$ |

given a maximum of 5 000 experiments for each configuration campaign. Candidates are evaluated according to the hypervolume metric, with reference points computed on-the-fly, and are discarded based on Friedman's non-parametric rank sum test using the default configurations of Irace.

## 7.3 Assessing the effectiveness of pheromone transfer proposals

The experiments discussed in the Chapter 6 showed that, when properly configured and in the absence of local search, the pheromone transfer approach used by P-ACO is particularly effective. Conversely, since $\mathcal{MM}$AS was proposed for static optimization, it fails to display competitive performance after the first problem change. Here, we assessed the effectiveness of some of the pheromone transfer proposals reviewed in Chapter 3 when coupled with $\mathcal{MM}$AS. Initially, we investigated the benefits of adding to $\mathcal{MM}$AS the pheromone transfer approach from P-ACO, while isolating the effects of parameter configuration and local search. Next, we compared several pheromone transfer approaches previously discussed, once again isolating the effects of local search. Finally, we assessed the combination of transfer approaches and compared the best-performing variant to P-ACO.

### 7.3.1 Assessing pheromone transfer through reset

As discussed in Chapter 3, P-ACO originally proposed that the pheromone information on edges $V_{s-1}/D_s$ should be (re)set to $\tau_0$ after each environment change, regulated by a forgetting parameter $\gamma \in [0, 1]$. In the context of $\mathcal{MM}$AS, this translated into (re)setting the pheromone information to $\tau_{max}$, since this was the initial pheromone value it adopts.

For the set of experiments conducted here, we compared three variants of $\mathcal{MM}$AS. The first version was the one we configured in the previous Chapter 6 (*mmasTuned*), which did not use any pheromone transfer mechanism, and served as a baseline. The second variant used the same parameter settings of the first one, but adopted the pheromone reset approach (*mmasTreset*).[2] Finally, the third version also adopted pheromone reset, but was re-configured by Irace (*mmasResetT*).

Our rationale was that the addition of pheromone transfer mechanisms could lead to interactions with other algorithmic components of $\mathcal{MM}$AS and thus required a

[2]Since the pheromone reset introduced the forgetting parameter $\gamma$, we have only configured this parameter.

Table 7.2: Parameters selected by Irace for $\mathcal{MM}$AS variants using different pheromone transfer approaches. Settings specific to runs with local search are indicated by the *LS* suffix.

| | $\mathcal{MM}$AS | | | | *reset* | *multi* |
|---|---|---|---|---|---|---|
| *Variant* | $m$ | $\alpha$ | $\beta$ | $\rho$ | $\gamma$ | $q_0$ |
| *mmasTreset* | 96* | 1* | 5* | 0.6* | 0.8 | – |
| *mmasTresetLS* | 9* | 1* | 1* | 0.4* | 0.7 | – |
| *mmasResetT* | 95 | 1 | 5 | 0.4 | 0.9 | – |
| *mmasResetTLS* | 9 | 1 | 2 | 0.4 | 0.6 | – |
| *mmasRestartT* | 96 | 1 | 5 | 0.3 | – | – |
| *mmasRestartTLS* | 6 | 1 | 1 | 0.1 | – | – |
| *mmasMultiT* | 91 | 1 | 2 | 0.3 | – | 0.3 |
| *mmasMultiTLS* | 9 | 1 | 1 | 0.8 | – | 0.3 |
| *mmasMultiRestartT* | 98 | 1 | 5 | 0.4 | – | 0.4 |
| *mmasMultiResetT* | 95 | 1 | 5 | 0.3 | 0.4 | 0.2 |

* Settings reused from the experiments conducted in the previous section.

different parameter configuration. Configured settings for this set of experiments are given in Table 7.2 (top rows). Like before, experiments were run isolating the effects of local search, and the *LS* suffix was added to configurations where 2-opt local search was adopted.

In the following, we discussed the most important insights we observed from our analysis, with the help of the SQT plots given in Figure 7.1. In particular, the plots depicted the anytime performance of the three different $\mathcal{MM}$AS variants we considered when run on a TSPLIB instance with (left) and without (right) local search, on the $\xi = 40\%$, $f = 10$ scenario.

- **Parameter settings**

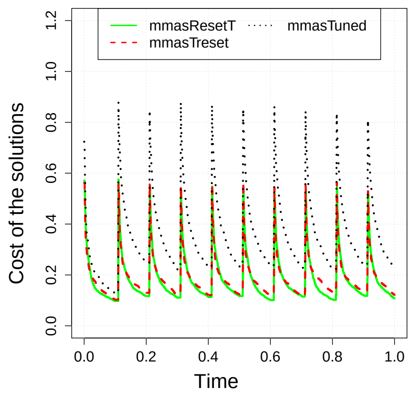
  The only parameters in which the settings of *mmasTreset* and *mmasResetT* differ were the evaporation ($\rho$) and the forgetting ($\gamma$) rates. The similarity also held between the variants that used local search (*mmasTresetLS* and *mmasResetTLS*), though this time it was $\beta$ rather than $\rho$ that was changed by Irace.

  From an algorithmic point of view, the most likely explanation for these changes concerned the absence of a pheromone transfer mechanism in the original $\mathcal{MM}$AS, which was compensated by Irace with an increased $\rho$ value. Concerning performance, both $\mathcal{MM}$AS variants that adopted the pheromone transfer mechanism from P-ACO perform similarly, although differences were mostly observed in favor of *mmasResetT*.

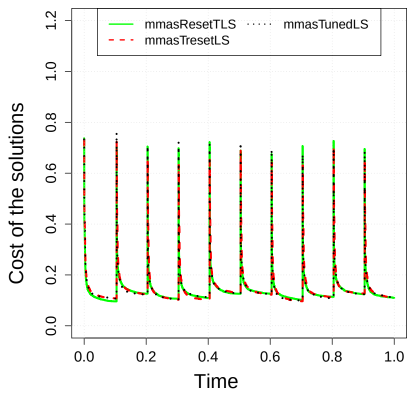
- **Improvements over the original $\mathcal{MM}$AS**

  Figure 7.1 (left) demonstrated the significant performance improvements provided by pheromone transfer through reset. In particular, the transfer approach helped address the most concerning issue with the original $\mathcal{MM}$AS, in that the performance of the variants were consistent throughout the run. More importantly, the benefits of transfer were observed for all instance sets, scenarios, and parameter configurations, as shown in the rank sum analysis given in Table 7.3.

  Conversely, the effects pheromone transfer mechanism were greatly reduced in the presence of local search, as shown in Figure 7.1 (right). In fact, it was often



(a) $\xi$=40, $f$=10



(b) $\xi$=40, $f$=10

Figure 7.1: SQT plots depicting the anytime performance of three different $\mathcal{MM}$AS versions, run on TSPLIB instance pr2392 with (right) and without (left) local search, using different parameter configurations on the $\xi = 40$, $f = 10$ scenario.

possible to identify performance differences in favor of the original $\mathcal{MM}$AS over the variants that adopted pheromone transfer. This was an important finding that corroborated our claim that algorithmic components proposed for dynamic optimization algorithms should be carefully assessed, in particular as to their possible interactions with other relevant components algorithms might use, such as local search.

## 7.3.2 Comparing pheromone transfer approaches

The insights produced in the previous section demonstrate that pheromone transfer approaches may interact with other algorithmic components, and hence reconfiguring variants is indicated. In this section, we compared three $\mathcal{MM}$AS variants, differing as to the transfer approach they adopt. The first variant is *mmasResetT*, which we preliminarily assessed in the previous section, and served as baseline. The second variant (*mmasRestartT*) restarted the pheromone trails of edges $V_{s-1}/D_s$ to $\tau_{max}$, completely forgetting the deposits from previous environments.[3] Finally, the third variant (*mmasMultiT*) follows the multi-caste approach (Melo et al., 2013) discussed in Chapter 3. In our work, ants were split into two groups of equal size $\lfloor m/2 \rfloor$. The first group of ants constructed solutions using probability $q_0$, whereas the second group used probability $1 - q_0$. Thus, depending on how $q_0$ is set, half of the ants would search favoring exploitation, with the other half favoring exploration.

- **Parameter settings**

  Configured settings for this set of experiments were given in Table 7.2 (middle rows). The different pheromone transfer strategies produced little effect on the number of ants ($m$) and importance of the pheromone information ($\alpha$). Indeed,

[3]Notice that the *restart* variant was equivalent to configuring parameter $\gamma$ to its maximum value in the *reset* variant.

the only noticeable differences between settings used by the variants concerned the importance of heuristic information ($\beta$) and the evaporation rate ($\rho$). However, no overall pattern could be easily observed, although it was remarkable that the variants that used local search presented so contrasting $\rho$ values. In particular, the very low $\rho$ value adopted by *mmasRestartT* was probably an effect of its strong forgetting behavior. Conversely, the very high $\rho$ value adopted by *mmasMultiT* was likely induced from not having an explicit transfer mechanism.

- **Frequency of change and instance characteristics**

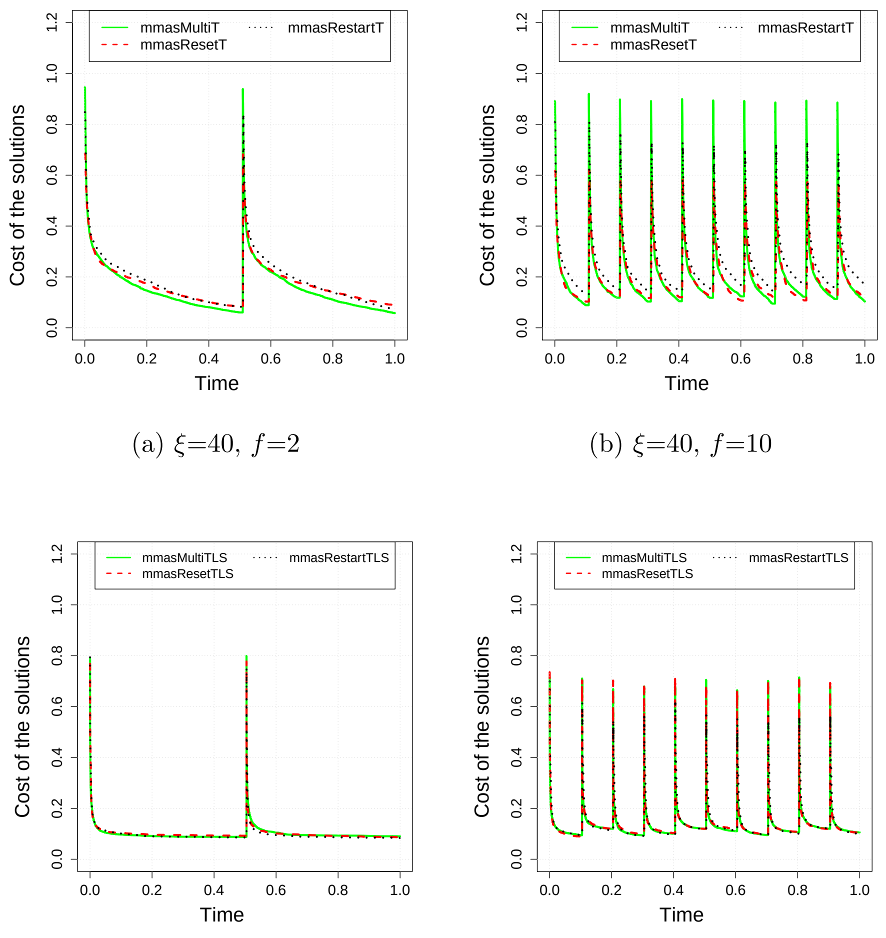
  The performance benefits provided by the different pheromone transfer approaches assessed in this section vary considerably as a function of the dynamic and structural characteristics of scenarios and instances. In general, the restart approach leaded to less benefits than reset regardless of set or scenario, as illustrated in Figure 7.2 (top). By contrast, the multi-caste approach leads to the most significant improvements only on RUE instances when $f = 2$, as shown on Figure 7.2 (top left). In more detail, when we considered TSPLIB instances (for all scenarios) or even RUE instances for $f = 10$ scenarios, adopting multiple castes was the approach that brought less performance benefits among all transfer approaches compared. This was corroborated by the analysis provided in Table 7.3, where *mmasResetT* was the best-performing variant. Regarding the frequency of change, the reduced benefits from multiple castes when $f = 10$ are likely related to the split computational resources among ant groups, given that the runtime available for each slot was much smaller than when $f = 2$. Concerning instance characteristics, the experiments in Section 4.3.2 had already indicated that $\mathcal{MM}$AS need more runtime for reoptimizing TSPLIB instances than for RUE instances, which became an issue for the multi-caste variant.

- **Local search**

  As in the preliminary assessment of pheromone transfer though reset, the presence of local search rendered the benefits from the transfer approaches minimal. This was illustrated in Figure 7.2 (bottom), where we observed that this performance similarity between approaches held whatever the frequency of change. Moreover, it was not possible to observe differences between transfer approaches whatever the instance set and degree of dynamism. Yet, when we aggregated over all runs considered, these small differences accumulated once again in favor of *mmasResetT*.

### 7.3.3 Combining pheromone transfer approaches

The assessment of pheromone transfer approaches produced two major insights. First, selecting a single pheromone transfer approach depended on the experimental factors adopted. Second, in the presence of local search, the benefits from all approaches considered became minimal. In this section, we assessed the combination of pheromone transfer approaches, to see if it was possible to combine their advantages into a single variant. In more detail, the multi-caste approach only affected the solution construction rule. This way, the pheromone update behavior of the



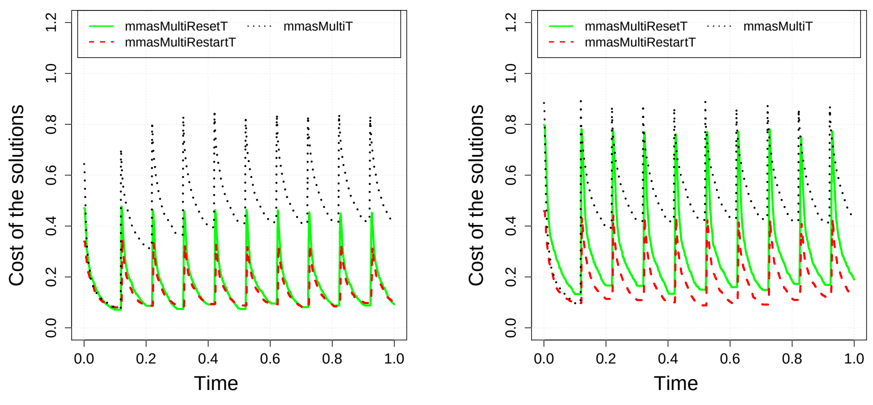
(a) $\xi$=40, $f$=2 (b) $\xi$=40, $f$=10

(c) $\xi$=40, $f$=2 (d) $\xi$=40, $f$=10

Figure 7.2: SQT plot depicting the anytime performance of $\mathcal{MM}$AS with three different versions of pheromone update mechanisms, run on TSPLIB instance pr2392 without local search, using same parameter configurations and different dynamic scenarios.

variants investigated in this section during a given environment was identical to the original $\mathcal{MM}$AS. Only when transitioning between environments was that the variants chose to either partially (*mmasMultiResetT*) or entirely (*mmasMultiRestartT*) forgot the pheromone information from previous environments. Furthermore, we remarked that we restricted our investigation in this section to experiments without local search.

Table 7.2 (bottom rows) showed the configured settings for the variants we considered in these experiments. The first variant was *mmasMultiT*, preliminarily assessed in the previous section, which we used as baseline. The second and third variants followed the multi-caste approach from the first variant, but explicitly promoted pheromone transfer through reset (*mmasMultiResetT*) or restart (*mmasMultiRestartT*). Overall, all three variants presented similar configuration, the strongest exception being parameter $\beta$, which was significantly increased for the novel variants.



(a) $\xi$=20, $f$=10 (b) $\xi$=80, $f$=10

Figure 7.3: SQT plots depicting the anytime performance of three different $\mathcal{MM}$AS variants, run on RUE instance 3002 using different parameter configurations on $f = 10$ scenarios, with $\xi = 20\%$ (left) and $\xi = 80\%$ (right) .

In a sense, $\mathcal{MM}$AS was now allowed to be more greedy as the transfer mechanisms help balance its search.

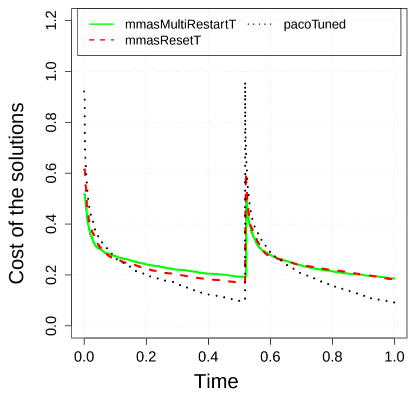
Figure 7.3 depicted SQT plots for experiments without local search, run on RUE instance 3002 on dynamic scenarios that presented frequency of change $f = 10$, but differ as to the degree of dynamism (left: $\xi = 20\%$; right: $\xi = 80\%$). The most evident observation was how the combination of transfer approaches improved over using multiple castes alone.

Two other observations stood out. First, it was rather interesting how the degree of dynamism affected the variants that did not adopt restart. In the case of *mmasMultiT*, its reoptimization in the initial environments was more effective when the degree of dynamism was lower. In the case of *mmasMultiResetT*, this loss in performance is observed across all environments, with a competitive performance on $\xi = 20\%$ scenarios contrasting with a less competitive performance on $\xi = 80\%$ scenarios.

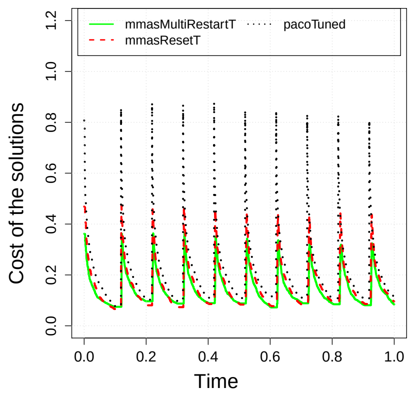
Second, *mmasMultiRestartT* was able to improve over *mmasMultiResetT*, specially for larger RUE instance sizes. Altogether, these findings were an excellent evidence of algorithmic component interaction, given that in the previous set of experiments *mmasResetT* had outperformed both restart and multi-caste for a significant number of experimental factors.

### 7.3.4 Comparing $\mathcal{MM}$AS variants with P-ACO

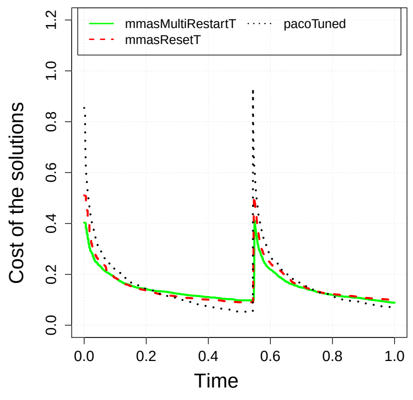
We concluded our investigation with a comparison of the best-performing algorithms and variants identified in this work. Namely, in this section we compared (i) P-ACO (*pacoTuned*); (ii) $\mathcal{MM}$AS coupled with pheromone transfer through reset (*mmasResetT*), and; (iii) $\mathcal{MM}$AS using the multi-settings approach coupled with pheromone transfer through restart (*mmasMultiRestartT*). Note that these experiments did not consider local search, as none of the algorithms and variants considered



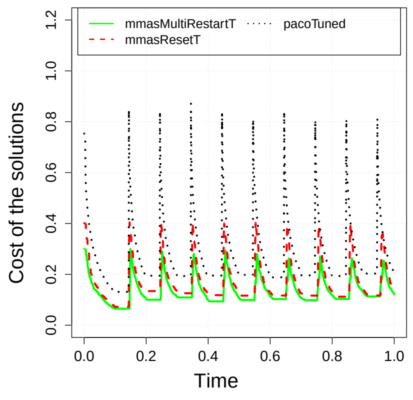
(a) $\xi$=40, $f$=2



(b) $\xi$=40, $f$=10



(c) $\xi$=40, $f$=2



(d) $\xi$=40, $f$=10

Figure 7.4: SQT plot depicting the anytime performance of best performing $\mathcal{MM}$AS versions and P-ACO, run on RUE instance 3005 ( 7.4a, 7.4b) and on TSPLIB instance pr2392 ( 7.4c, 7.4d) without local search, using different parameter configurations and different dynamic scenarios. Left: scenario $\xi 40 f 2$. Right: scenario $\xi 40 f 10$.

here were competitive with (P-ACO) or able to improve significantly over (*mmasResetT* and *mmasMultiRestartT*) the original $\mathcal{MM}$AS coupled with local search. Our main objective here was then to measure the relative performance of the $\mathcal{MM}$AS variants w.r.t. P-ACO when local search was not adopted.

Figure 7.4 showed SQT plots depicting runs without local search on RUE instances 3002 (top) and 4002 (bottom), on scenarios that presented degree of dynamism $\xi = 40\%$, but vary as to frequency of change (left: $f = 2$; right: $f = 10$). These plots helped illustrate the two more relevant factors that affect results, as follows. First, the relative performance of the $\mathcal{MM}$AS variants w.r.t. P-ACO was strongly affected by the frequency of change. Specifically, Figure 7.4 (left) showed that P-ACO outperformed all $\mathcal{MM}$AS variants when $f = 2$. Conversely, we see from Figure 7.4 (right) that the $\mathcal{MM}$AS variants outperformed P-ACO when there was an increase in the frequency of change. This was a rather remarkable result that

Table 7.3: Statistical analysis of the experiments conducted in this section.

| *Section* | *Setup* | $\Delta R$ | *Rank sums* | | |
|---|---|---|---|---|---|
| 7.3.1 | **Without LS** | 17.51 | ***mmasResetT*** | *mmasTreset* (65) | *mmasTuned* (178) |
| | **With LS** | 27.65 | ***mmasTunedLS*** | ***mmasResetTLS*** (7) | *mmasTresetLS* (122) |
| 7.3.2 | **Without LS** | 29.62 | ***mmasResetT*** | *mmasMultiT* (98) | *mmasRestartT* (115) |
| | **With LS** | 31.51 | ***mmasResetTLS*** | *mmasMultiTLS* (34) | *mmasRestartTLS* (107) |
| 7.3.3 | **Without LS** | 35.69 | ***mmasMultiRestartT*** | ***mmasMultiResetT*** (10) | *mmasMultiT* (56) |
| 7.3.4 | **Without LS** | 36.31 | ***pacoTuned*** | ***mmasResetT*** (28) | *mmasMultiRestartT* (47) |

confirmed that $\mathcal{MM}$AS could be competitive in the context of dynamic optimization even in the absence of local search procedures.

The second factor that affect results was the number of customers. In particular, we observed that the instance size played a more relevant role than the instance structural characteristics in this particular assessment. For this reason, we included both a smaller (top) and a larger (bottom) RUE instances, with the smaller instance being representative of the results for TSPLIB. When $f = 2$, P-ACO improved over the $\mathcal{MM}$AS variants by a larger margin on smaller instances (Figure 7.4a) than on larger instances (Figure 7.4c). Conversely, when $f = 10$ P-ACO was only competitive on smaller instances (Figure 7.4b), and was worse than the $\mathcal{MM}$AS variants by a significant gap on larger instances (Figure 7.4d). When we look at the overall picture given in Table 7.3, we saw that *pacoTuned* and *mmasResetT* were considered equivalent.

Altogether, these experimental factors suggested that the results from this comparison may have been influenced by the configuration setup adopted. More precisely, we decided to use a single configuration for multiple experimental scenarios and benchmark instance sets. In addition, the TSPLIB instances we selected were smaller than the RUE instances we created. Balancing instance sizes among benchmark sets and adopting configurations specific to each scenario and set would likely help further understand the performances of these algorithms and variants. However, we made significant progress through this performance assessment, in that many factors that have been previously overlooked led to important insights, such as the effects of parameter configuration and the interactions between algorithmic components, such as pheromone transfer and local search.

# Chapter 8

# Conclusion

Efforts have been dedicated over the years to develop high-quality algorithms when applied to DCOPs. However, many issues were identified from the literature review about how to implement and/or improve algorithms when applied to this problem class. As a main consequence, experiments could conduct researchers to erroneous analyses and conclusions. Thus, the main purpose of this research was the development of an empirical way to address all these issues. They were: (i) the use of automatic tuning to properly adapt the algorithm's parameter when tackling DCOPs; (ii) the use of local search to improve algorithms performance and (iii) the application of the hypervolume as a unique and complete performance metric. The ACO and the TSPDDs were chosen as algorithm and problem benchmark, respectively.

Among the most relevant ACO algorithms proposed in the literature, there are those that re-evaluate the pheromone information deposited on the solution components affected by problem changes. The best known of these algorithms was P-ACO (Guntsch e Middendorf, 2002), which was based on a transient pheromone memory that quickly adapts exploiting the most recent solutions found. Another major share of the ACO algorithms applied to dynamic optimization literature focused on extending $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS (Stützle e Hoos, 2000)), given its excellent performance on static optimization. Concerning the TSP dynamic variation, the TSPDDs, its simplicity of implementation, close relation to real world problems and influence on testing new algorithms were the main facts that defined the use of this problem class in our research.

The experiments presented in this research resulted in the publication of one book chapter, three conference papers and two journal papers. The main contributions of each publication were summarized as follows. The first publication was dedicated to a review about ACO metaheuristic, its algorihtms and their application on COPs in order to improve the ACO knowledge background that was going to be needed in this research (Dorigo et al., 2011). This publication was followed by the work that highlighted the issue faced by researchers when applying manual tuning and the influence of local search (Oliveira et al., 2011a). There, P-ACO algorithm was chosen to be applied, and two problems were used as benchmark: the TSP and the Quadratic Assigned Problem (QAP). Here we concentrated our efforts only on the TSP. From the insights obtained after the analysis of the experiments conducted, we highlighted that the usage or not of a local search had a strong impact on parameters settings and pheromone update for P-ACO applied to the TSP.

After this publication, the TSP was replaced by the TSPDDs, which was the main problem chosen to be applied to this research. The TSPDDs had not been applied in the first place due to the fact that we would like to have the first insights about the importance of automatic tuning from a problem that had been extensively studied in the literature. In this way, we would be able to see whether our conclusions corroborated with the ones already presented in the literature so far. This experience would serve as a knowledge source to move from static to dynamic COPs. This replacement resulted in another publication (Oliveira et al., 2019) and in a journal paper (da Silva et al., 2020). From the results obtained, we could see how dynamic optimization problems demand algorithms engineered to quickly produce high-quality solutions, particularly after problem data changed. This dichotomy would be more accurately addressed when formulated as a bi-objective optimization problem, where solution quality and runtime were the most prototypical objective examples. Through the extension of this formulation and application of the concept of anytime behavior to a dynamic optimizer, we were able to use the hypervolume to compare any number of algorithms and, under certain circumstances, benefit from the Pareto-compliance property of this indicator. Together, these characteristics greatly improved over previous measures adopted in the dynamic optimization Literature. To empirically evaluate our proposed approach, we also conducted an experimental study on the performance of different variants of the P-ACO algorithm run on the TSPDDs. Surprisingly, the variant configured for static optimization performed better than the one configured for dynamic optimization. More importantly, once again, we saw that local search was a critical component even in the context of dynamic optimization, leading to the best anytime behavior in most of the experiments conducted.

The methodology presented was then successfully applied in the field of multi-criteria decision making (MCDM) to evaluate the impact of social distancing and mobility evaluation for human mobility during the first COVID-19 lock down considering (i) multiple categories for a given time period and (ii) multiple categories over multiple time periods. We empirically demonstrated these approaches by conducting both a region- and country-level analysis, comparing some of the most relevant outbreak examples from different continents.

After the setup of a proper way to evaluate end compare algorithms' performance when applied to TSPDDs, we were able to define an automatic way to set up algorithms parameters and improve ACO main procedures. Thus we had proposed the use of the Irace tool to tune all $\mathcal{MM}$AS and P-ACO parameters. By using the Irace tool we were also able to include the hypervolume measure as a methodology for selecting the most promising parameters during the tuning process. A new parameter setting was defined for both $\mathcal{MM}$AS and P-ACO. Latter, improvements were proposed to $\mathcal{MM}$AS generating $\mathcal{MM}$AS extensions which were also able to be tuned due to the velocity of tuning algorithms automatically. In this way, we had assessed P-ACO and many such *pheromone transfer* approaches through a rigorous computational study on a dynamic variant of the traveling salesman problem (TSP). All possible contributions concerning the automatic tuning tool and $\mathcal{MM}$AS improvements were then assigned to a journal paper (Oliveira et al., 2021).

In summary, our work produced, at least, three major contributions. First, we proposed an automatic configuration approach for dynamic optimization, where we use the hypervolume as a metric to evaluate the anytime behavior of algorithms. Second, we compared P-ACO to $\mathcal{MM}$AS, one of the best-performing ACO algorithms for the static TSP, isolating the effects of parameter settings and local search. In the absence of local search, as traditionally observed in the dynamic optimization literature, P-ACO outperforms $\mathcal{MM}$AS by a large margin. Yet, when the local search is adopted, $\mathcal{MM}$AS consistently outperforms P-ACO on all experimental scenarios considered. Third, we investigated the benefits of ACO adaptations for dynamic optimization, observing that their benefit is considerably reduced in the presence of local search. However, pheromone transfer approaches render $\mathcal{MM}$AS more effective than P-ACO for a large part of the experimental scenarios where local search is not adopted.

The insights produced in this investigation opened several possible future work directions. Concerning the automatic configuration of anytime behavior for dynamic optimization, results revealed the importance of both instance characteristics and size, indicating better results could likely be obtained from isolating these factors for configuration. More importantly, given the overhead required by offline configuration, it would be paramount to assemble a repository of configured settings for multiple algorithms, benchmark sets, and experimental scenarios. In addition, we have only just started taking advantage of the theoretical benefits of hypervolume. For instance, it remains an open question whether the Pareto-compliant properties of the hypervolume when measuring the anytime behavior within a single environment could also be extended for definite conclusions about the whole run.

Regarding the comparison between $\mathcal{MM}$AS and P-ACO, we observed that conclusions are strongly dependent on local search. Yet, we considered a single local search operator, which is neither the most efficient nor the most effective for the static TSP. An extended investigation on operators possibly better suited for dynamic optimization would likely indicate a means to reduce overhead while preserving the performance benefits. Conversely, in the absence of local search, it was not possible to identify a single best-performing algorithm for all scenarios. Nonetheless, our work has paved the way for other high-performing ACO algorithms to be extended to dynamic optimization.

# Bibliography

Adenso-Diaz, Belarmino e Laguna, Manuel. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations research*, v. 54, n. 1, p. 99–114.

Alba, Enrique e Sarasola, Briseida. (2010)a. ABC, a new performance tool for algorithms solving dynamic optimization problems. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2010), Barcelona, Spain, 18-23 July 2010*, p. 1–7, (2010)a. doi: 10.1109/CEC.2010.5586406.

Alba, Enrique e Sarasola, Briseida. (2010)b. Measuring fitness degradation in dynamic optimization problems. Di Chio, Cecilia; Cagnoni, Stefano; Cotta, Carlos; Ebner, Marc; Ekárt, Anikó; Esparcia-Alcazar, Anna I.; Goh, Chi-Keong; Merelo, Juan J.; Neri, Ferrante; Preuß, Mike; Togelius, Julian e Yannakakis, Georgios N., editors, *Proceedings of the European Conference on the Applications of Evolutionary Computation: Applications of Evolutionary Computation (EvoApplications 2010), Part I*, volume 6024, p. 572–581, Berlin, Heidelberg. Springer.

Ben-Romdhane, Hajer; Alba, Enrique e Krichen, Saoussen. (2013). Best practices in measuring algorithm performance for dynamic optimization problems. *Soft Computing*, v. 17, n. 6, p. 1005–1017. doi: 10.1007/s00500-013-0989-7.

Benedettini, S.; Blum, C. e Roli, A. (2010). A randomized iterated greedy algorithm for the founder sequence reconstruction problem. *Learning and Intelligent Optimization, 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers*, p. 37–51, (2010).

Bezerra, Leonardo C. T. *A component-wise approach to multi-objective evolutionary algorithms: from flexible frameworks to automatic design.* PhD thesis, IRIDIA, CoDE, Université Libre de Bruxelles, (2016).

Birattari, Mauro. (2009). F-race for tuning metaheuristics. *Tuning metaheuristics*, p. 85–115. Springer.

Birattari, Mauro; Stützle, Thomas; Paquete, Luis e Varrentrapp, Klaus. (2002). A racing algorithm for configuring metaheuristics. Langdon, W. B.; Cantu-Paz, E.; Mathias, K.; Roy, R.; Davis, D.; Poli, R.; Balakrishnan, K.; Honavar, V.; Rudolph, G.; Wegener, J.; Bull, L.; Potter, M. A.; Schultz, A. C.; Miller, J. F.; Burke, E. e Jonoska, N., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2002)*, volume 2, (2002).

Birattari, Mauro; Yuan, Zhi; Balaprakash, Prasanna e Stützle, Thomas. (2010). *F-Race and iterated F-Race: An overview*, p. 311–336. Springer. doi: 10.1007/978-3-642-02538-9_13.

Blum, Christian; Blesa, María J. e López-Ibáñez, Manuel. (2009). Beam search for the longest common subsequence problem. *Computers & Operations Research*, v. 36, n. 12, p. 3178–3186.

Branke, Jürgen; Salihoğlu, Erdem e Uyar, Şima. (2005). Towards an analysis of dynamic environments. Beyer, Hans-Georg e O'Reilly, Una-May, editors, *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'05), Washington DC, USA, June 25-29, 2005*, p. 1433–1440, New York, NY, USA. ACM. doi: 10.1145/1068009.1068237.

Conover, William Jay e Conover, William Jay. (1980). *Practical nonparametric statistics*. Wiley New York.

Coy, Steven P; Golden, Bruce L; Runger, George C e Wasil, Edward A. (2001). Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, v. 7, n. 1, p. 77–97.

Cruz, Carlos; González, Juan R. e Pelta, David A. (2011). Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*, v. 15, n. 7, p. 1427–1448. doi: 10.1007/s00500-010-0681-0.

daSilva, Gabriela Cavalcante; Oliveira, Sabrina; Wanner, Elizabeth F e Bezerra, Leonardo C. T. Google covid-19 community mobility reports: insights from multi-criteria decision making, (2020). URL `https://arxiv.org/abs/2009.10648`.

Dorigo, M. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, (1992).

Dorigo, M. e Stützle, T. (2004). *Ant Colony Optimization*. MIT Press, Cambridge, MA.

Dorigo, Marco e Gambardella, Luca Maria. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, v. 1, n. 1, p. 53–66.

Dorigo, Marco; Maniezzo, Vittorio e Colorni, Alberto. June(1991). Positive feedback as a search strategy. Report 91-016, Laboratorio di Calcolatori, Dipartimento di Elettronica, Politecnico di Milano.

Dorigo, Marco; Maniezzo, Vittorio e Colorni, Alberto. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, v. 26, n. 1, p. 29–41.

Dorigo, Marco; Montes de Oca, Marco A.; Oliveira, Sabrina e Stützle, Thomas. (2011). Ant colony optimization. Cochran, James J.; Cox, Louis A.; Keskinocak, Pinar; Kharoufeh, Jeffrey P. e Smith, J. Cole, editors, *Wiley Encyclopedia of Operations Research and Management Science*, volume 1, p. 114–125. John Wiley & Sons, Inc.

Eyckelhof, Casper Joost e Snoek, Marko. (2002). Ant systems for a dynamic TSP: Ants caught in a traffic jam. Dorigo, Marco; Caro, Gianni Di e Sampels, Michael, editors, *Proceedings of the Third International Workshop on Ant Algorithms (ANTS 2002), Brussels, Belgium, September 12-14 2002*, volume 2463 of *Lecture Notes in Computer Science*, p. 88–99, Berlin, Heidelberg. Springer. doi: 10.1007/3-540-45724-0_8.

Gambardella, Luca Maria e Dorigo, Marco. (1996). Solving symmetric and asymmetric tsps by ant colonies. *Proceedings of IEEE international conference on evolutionary computation*, p. 622–627. IEEE, (1996).

Gendreau, Michel; Hertz, Alain; Laporte, Gilbert e Stan, Mihnea. (1998). A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, v. 46, n. 3, p. 330–335. URL `https://EconPapers.repec.org/RePEc:inm:oropre:v:46:y:1998:i:3:p:330-335`.

Guntsch, M. Ant Algorithms in Stochastic and Multi-Criteria Environments. PhD thesis, Universität Fridericiana zu Karlsruhe, (2004).

Guntsch, M. e Middendorf, M. (2001). Pheromone modification strategies for ant algorithms applied to dynamic TSP. Boers, E. J. W. e others,, editors, *EvoWorkshops 2001*, volume 2037 of *LNCS*, p. 213–222. Springer Verlag, Berlin, Germany.

Guntsch, Michael e Middendorf, Martin. (2002). Applying population based ACO to dynamic optimization problems. Dorigo, Marco; Caro, Gianni Di e Sampels, Michael, editors, *Proceedings of the Third International Workshop on Ant Algorithms (ANTS 2002), Brussels, Belgium, September 12-14 2002*, volume 2463 of *Lecture Notes in Computer Science*, p. 111–122, Berlin, Heidelberg. Springer. doi: 10.1007/3-540-45724-010.

Guntsch, Michael; Middendorf, Martin e Schmeck, Hartmut. (2001). An ant colony optimization approach to dynamic TSP. Spector, Lee; Goodman, Erik D.; Wu, Annie; Langdon, W. B. e Voigt, Hans-Michael, editors, *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation (GECCO'01), San Francisco, California, USA, July 07-11, 2001*, p. 860–867, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Gutin, Gregory e Punnen, Abraham P. (2006). *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media.

Hoos, H. e Stützle, T. (2004). *Stochastic Local Search: Foundations and Applications.* Morgan Kaufmann Publishers, San Francisco, CA.

http://dimacs.rutgers.edu/archive/Challenges/TSP/,. 8th DIMACS Implementation Challenge: The Traveling Salesman Problem, (2018). .

Knowles, Joshua; Thiele, Lothar e Zitzler, Eckart. February(2006). A tutorial on the performance assessment of stochastic multiobjective optimizers. TIK-Report 214, TIK-ETH Zürich. Revised version.

López-Ibáñez, Manuel e Blum, Christian. 09(2010). Beam-aco for the travelling salesman problem with time windows. *Computers & OR*, v. 37, p. 1570–1583. doi: 10.1016/j.cor.2009.11.015.

López-Ibáñez, Manuel; Dubois-Lacoste, Jérémie; Pérez-Cácerez, Leslie; Stützle, Thomas e Birattari, Mauro. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, v. 3, p. 43–58. doi: 10.1016/j.orp.2016.09.002.

López-Ibánez, Manuel e Stützle, Thomas. (2014). Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research*, v. 235, n. 3, p. 569–582.

Mavrovouniotis, M.; Müller, F. M. e Yang, S. (2017). Ant colony optimization with local search for dynamic traveling salesman problems. *IEEE Transactions on Cybernetics*, v. 47, n. 7, p. 1743–1756. doi: 10.1109/TCYB.2016.2556742.

Mavrovouniotis, M. e Yang, S. (2015). Population-based incremental learning with immigrants schemes in changing environments. *Computational Intelligence, 2015 IEEE Symposium Series on*, p. 1444–1451, (2015).

Mavrovouniotis, M.; Yang, Shengxiang e Yao, Xin. Dec(2014). Multi-colony ant algorithms for the dynamic travelling salesman problem. *Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2014 IEEE Symposium on*, p. 9–16, Dec(2014). doi: 10.1109/CIDUE.2014.7007861.

Mavrovouniotis, Michalis; Bonilha, Iaê S; Müller, Felipe M; Ellinas, Georgios e Polycarpou, Marios. (2019). Effective aco-based memetic algorithms for symmetric and asymmetric dynamic changes. *2019 IEEE Congress on Evolutionary Computation (CEC)*, p. 2567–2574. IEEE, (2019).

Mavrovouniotis, Michalis; Li, Changhe e Yang, Shengxiang. (2017). A survey of swarm intelligence for dynamic optimization: Algorithms and applications. *Swarm and Evolutionary Computation*, v. 33, p. 1 – 17.

Mavrovouniotis, Michalis; Müller, Felipe Martins e Yang, Shengxiang. (2015). An ant colony optimization based memetic algorithm for the dynamic travelling salesman problem. *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, p. 49–56. ACM, (2015). doi: 10.1145/2739480.2754651.

Mavrovouniotis, Michalis e Yang, Shengxiang. (2010). Ant colony optimization with immigrants schemes in dynamic environments. Schaefer, Robert; Cotta, Carlos; Kolodziej, Joanna e Rudolph, Günter, editors, *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN XI), Kraków, Poland, September 11-15, 2010, Part II*, volume 6239 of *Lecture Notes in Computer Science*, p. 371–380, Berlin, Heidelberg. Springer. doi: 10.1007/978-3-642-15871-1_38.

Mavrovouniotis, Michalis e Yang, Shengxiang. jul(2011)a. A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. *Soft Comput.*, v. 15, n. 7, p. 1405–1425. ISSN 1432-7643. doi: 10.1007/s00500-010-0680-1.

Mavrovouniotis, Michalis e Yang, Shengxiang. (2011)b. Memory-based immigrants for ant colony optimization in changing environments. Di Chio, Cecilia; Cagnoni, Stefano; Cotta, Carlos; Ebner, Marc; Ekárt, Anikó; Esparcia-Alcázar, Anna I.; Merelo, Juan J.; Neri, Ferrante; Preuss, Mike; Richter, Hendrik; Togelius, Julian e Yannakakis, Georgios N., editors, *Proceedings of the European Conference on the Applications of Evolutionary Computation: Applications of Evolutionary Computation (EvoApplications 2011: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC), Torino, Italy, April 27-29, 2011, Part I*, volume 6624 of *Lecture Notes in Computer Science*, p. 324–333, Berlin, Heidelberg. Springer. doi: 10.1007/978-3-642-20525-5_33.

Mavrovouniotis, Michalis e Yang, Shengxiang. (2012). Ant colony optimization with memory-based immigrants for the dynamic vehicle routing problem. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2012), Brisbane, Australia, June 10-15, 2012*, p. 1–8, (2012). doi: 10.1109/CEC.2012.6252885.

Mavrovouniotis, Michalis e Yang, Shengxiang. (2013)a. Adapting the pheromone evaporation rate in dynamic routing problems. Esparcia-Alcázar, Anna Isabel, editor, *Proceedings of the 16th European Conference on the Applications of Evolutionary Computation (EvoApplications 2013), Vienna, Austria, April 3-5, 2013*, volume 7835 of *Lecture Notes in Computer Science*, p. 606–615, Berlin, Heidelberg. Springer. doi: 10.1007/978-3-642-37192-961.

Mavrovouniotis, Michalis e Yang, Shengxiang. (2013)b. Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors. *Applied Soft Computing*, v. 13, n. 10, p. 4023–4037. doi: 10.1016/j.asoc.2013.05.022.

Mavrovouniotis, Michalis e Yang, Shengxiang. (2014)a. Ant colony optimization with self-adaptive evaporation rate in dynamic environments. *Proceedings of the 2014 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE 2014), Orlando, FL, USA, December 9-12, 2014*, p. 47–54, (2014)a. doi: 10.1109/CIDUE.2014.7007866.

Mavrovouniotis, Michalis e Yang, Shengxiang. (2014)b. Elitism-based immigrants for ant colony optimization in dynamic environments: Adapting the replacement rate. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2014, Beijing, China, July 6-11, 2014*, p. 1752–1759, (2014)b. doi: 10.1109/CEC.2014.6900482.

Mavrovouniotis, Michalis e Yang, Shengxiang. (2018). Ant colony optimization for dynamic combinatorial optimization problems. *Swarm Intelligence*, v. 1, p. 121–142.

Mavrovouniotis, Michalis; Yang, Shengxiang; Van, Mien; Li, Changhe e Polycarpou, Marios. (2020). Ant colony optimization algorithms for dynamic optimization: A case study of the dynamic travelling salesperson problem [research frontier]. *IEEE Computational Intelligence Magazine*, v. 15, n. 1, p. 52–63.

Melo, Leonor Albuquerque; Pereira, Francisco Baptista e Costa, Ernesto. (2013). Multi-caste ant colony algorithm for the dynamic traveling salesperson problem. *Proceedings of the 11th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2013), Lausanne, Switzerland, April 4-6, 2013*, volume 7824 of *Lecture Notes in Computer Science*, p. 179–188, Berlin, Heidelberg. Springer Berlin Heidelberg. doi: 10.1007/978-3-642-37213-1_19.

Monnot, Jérôme e Toulouse, Sophie. (2014). *The traveling salesman problem and its variations*, p. 173–214. Wiley Online Library, 2th edição.

Mori, Naoki; Kita, Hajime e Nishikawa, Yoshikazu. (2001). Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. *Transactions of the Institute of Systems, Control and Information Engineers*, v. 14, n. 1, p. 33–41. doi: 10.5687/iscie.14.33.

Morrison, Ronald W. (2003). Performance measurement in dynamic environments. Branke, Jürgen, editor, *Proceedings of the Workshop on Evolutionary Algorithms for Dynamic Optimization Problems (EvoDOP-2003) held in conjunction with the Genetic and Evolutionary Computation Conference (GECCO-2003), 12 July 2003, Chicago, USA*, p. 5–8, (2003).

Mosayebi, Mohsen; Sodhi, Manbir e Wettergren, Thomas A. (2021). The traveling salesman problem with job-times (tspj). *Computers & Operations Research*, v. 129, p. 105226.

Nguyen, Trung Thanh; Yang, Shengxiang e Branke, Jüergen. (2012)a. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, v. 6, p. 1–24. doi: 10.1016/j.swevo.2012.05.001.

Nguyen, Trung Thanh; Yang, Shengxiang e Branke, Jüergen. (2012)b. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, v. 6, p. 1–24. doi: 10.1016/j.swevo.2012.05.001.

Nguyen, Trung Thanh e Yao, Xin. (2012). Continuous dynamic constrained optimization – The Challenges. *IEEE Trans. Evolutionary Computation*, v. 16, n. 6, p. 769–786. doi: 10.1109/TEVC.2011.2180533.

Oliveira, S.; Hussin, M. S.; Roli, A.; Dorigo, M. e Stützle, T. (2017). Analysis of the population-based ant colony optimization algorithm for the tsp and the qap. *2017 IEEE Congress on Evolutionary Computation (CEC)*, p. 1734–1741, (2017).

Oliveira, Sabrina; Bezerra, Leonardo; Stützle, Thomas; Dorigo, Marco; Wanner, Elizabeth e de Souza, Sérgio R. 05(2021). A computational study on ant colony optimization for the traveling salesman problem with dynamic demands. *Computers & Operations Research*, v. 135, p. 105359. doi: 10.1016/j.cor.2021.105359.

Oliveira, Sabrina; Hussin, Mohamed Saifullah; Stützle, Thomas; Roli, Andrea e Dorigo, Marco. (2011)a. A Detailed Analysis of the Population-Based Ant Colony Optimization Algorithm for the TSP and the QAP. Relatório Técnico TR/IRIDIA/2011-006, IRIDIA, Université Libre de Bruxelles.

Oliveira, Sabrina; Wanner, Elizabeth F.; de Souza, Sérgio R.; Bezerra, Leonardo C. T. e Stützle, Thomas. (2019). The hypervolume indicator as a performance measure in dynamic optimization. Deb, Kalyanmoy; Goodman, Erik; Coello Coello, Carlos A.; Klamroth, Kathrin; Miettinen, Kaisa; Mostaghim, Sanaz e Reed, Patrick, editors, *Proceedings of the 10th International Conference on Evolutionary Multi-Criterion Optimization (EMO 2019), East Lansing, MI, USA, March 10-13, 2019*, volume 11411 of *Lecture Notes in Computer Science*, p. 319–331, Cham. Springer. doi: 10.1007/978-3-030-12598-1.

Oliveira, Sabrina M.; Hussin, Mohamed Saifullah; Stützle, Thomas; Roli, Andrea e Dorigo, Marco. (2011)b. A detailed analysis of the population-based ant colony optimization algorithm for the TSP and the QAP. *GECCO (Companion)'11*, (2011)b.

Pellegrini, Paola; Stützle, Thomas e Birattari, Mauro. (2010). Off-line vs. on-line tuning: A study on max-min ant system for the tsp. *International Conference on Swarm Intelligence*, p. 239–250. Springer, (2010).

Pina-Pardo, Juan C; Silva, Daniel F e Smith, Alice E. (2021). The traveling salesman problem with release dates and drone resupply. *Computers & Operations Research*, v. 129, p. 105170.

Psaraftis, Harilaos N.; Wen, Min e Kontovas, Christos A. (2016). Dynamic vehicle routing problems: Three decades and counting. *Networks*, v. 67, n. 1, p. 3–31. doi: 10.1002/net.21628.

Radulescu, Andreea; López-Ibáñez, Manuel e Stützle, Thomas. (2013). Automatically improving the anytime behaviour of multiobjective evolutionary algorithms. Purshouse, Robin C.; Fleming, Peter J.; Fonseca, Carlos M.; Greco, Salvatore e Shaw, Jane, editors, *Proceedings of the 7th International Conference on Evolutionary Multi-Criterion Optimization (EMO 2013), Sheffield, UK, March 19-22, 2013*, volume 7811 of *Lecture Notes in Computer Science*, p. 825–840, Berlin, Heidelberg. Springer.

Rand, William e Riolo, Rick L. (2005). 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 7th Annual Workshop on Genetic and Evolutionary Computation (GECCO'05), Washington DC, USA, June 25-26, 2005*, p. 32–38, New York, NY, USA. ACM. doi: 10.1145/1102256.1102263.

Reinelt, Gerhard. TSPLIB, (2008). http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

Ridge, Enda e Kudenko, Daniel. (2007). Tuning the performance of the mmas heuristic. *International Workshop on Engineering Stochastic Local Search Algorithms*, p. 46–60. Springer, (2007).

Rohlfshagen, Philipp e Yao, Xin. (2008). Attributes of dynamic combinatorial optimisation. Li, Xiaodong; Kirley, Michael; Zhang, Mengjie; Green, David; Ciesielski, Vic; Abbass, Hussein; Michalewicz, Zbigniew; Hendtlass, Tim; Deb, Kalyanmoy; Tan, Kay Chen; Branke, Jürgen e Shi, Yuhui, editors, *Simulated Evolution and Learning: 7th International Conference, SEAL 2008, Melbourne, Australia, December 7-10, 2008. Proceedings*, p. 442–451, Melbourne, Australia. Springer Berlin Heidelberg.

Sarasola, Briseida e Alba, Enrique. (2013). Quantitative performance measures for dynamic optimization problems. Alba, Enrique; Nakib, Amir e Siarry, Patrick, editors, *Metaheuristics for Dynamic Optimization*, volume 433 of *Studies in Computational Intelligence*, p. 17–33. Springer, Berlin, Heidelberg. doi: 10.1007/978-3-642-30665-5_2.

Schmitt, João P; Parpinelli, Rafael S e Baldo, Fabiano. (2019). Analysis of max-min ant system with local search applied to the asymmetric and dynamic travelling salesman problem with moving vehicle. *International Symposium on Experimental Algorithms*, p. 202–218. Springer, (2019).

Schmitt, João Pedro; Baldo, Fabiano e Parpinelli, Rafael Stubs. (2018). A max-min ant system with short-term memory applied to the dynamic and asymmetric traveling salesman problem. *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*, p. 1–6. IEEE, (2018).

Stützle, T. e Dorigo, M. (1999). ACO algorithms for the traveling salesman problem. Miettinen, K.; Mäkelä, M. M.; Neittaanmäki, P. e Périaux, J., editors, *Evolutionary Algorithms in Engineering and Computer Science*, p. 163–183. John Wiley & Sons, Chichester, UK.

Stützle, Thomas e Hoos, Holger. April 2-4(1998). Improvements on the ant-system: Introducing the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. *Proceedings of the Third International Conference on Artificial Neural Nets and Genetic Algorithms (ICANNGA 97)*, p. 245–249, Norwich, U.K. Springer Vienna.

Stützle, Thomas e Hoos, Holger H. (2000). $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, v. 16, n. 8, p. 889–914. doi: 10.1016/S0167-739X(00)00043-1.

Thalheim, Torsten; Merkle, Daniel e Middendorf, Martin. March 19 - 21(2008). A hybrid population based ACO algorithm for protein folding. Ao, S. I.; Castillo, Oscar; Douglas, Craig; Feng, David Dagan e Lee, Jeong-A, editors, *Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 (IMECS'08), Vol. I*, p. 200–205, Hong Kong.

Weicker, Karsten. (2002). Performance measures for dynamic environments. Merelo Guervós, Juan Julián; Adamidis, Panagiotis; Beyer, Hans-Georg; Martín, José Luis Fernández-Villacañas e Schwefel, Hans-Paul, editors, *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN VII), Granada, Spain, September 7-11, 2002*, volume 2439 of *Lecture Notes in Computer Science*, p. 64–76, Berlin Heidelberg. Springer-Verlag. doi: 10.1007/3-540-45712-7_7.

Xin Yu,; Ke Tang, e Xin Yao,. (2008). An immigrants scheme based on environmental information for genetic algorithms in changing environments. *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence) – CEC 2008, June 1-6, 2008, Hong Kong, China*, p. 1141–1147, (2008). doi: 10.1109/CEC.2008.4630940.

Yang, S.; Jiang, Y. e Nguyen, T. T. Oct(2013). Metaheuristics for dynamic combinatorial optimization problems. *IMA Journal of Management Mathematics*, v. 24, n. 4, p. 451–480. doi: 10.1093/imaman/dps021.

Yang, Shengxiang. June 25-29(2005). Memory-based immigrants for genetic algorithms in dynamic environments. *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO 2005)*, p. 1115–1122, New York, NY, USA. ACM. doi: 10.1145/1068009.1068196.

Yang, Shengxiang. (2007). Genetic algorithms with elitism-based immigrants for changing optimization problems. Giacobini, Mario, editor, *Proceedings of the Workshops on Applications of Evolutionary Computation (EvoWorkshops 2007) EvoCoMnet,EvoFIN, EvoIASP,EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog, Valencia, Spain, April 11-13, 2007*, volume 4448 of *Lecture Notes in Computer Science*, p. 627–636, Berlin, Heidelberg. Springer. doi: 10.1007/978-3-540-71805-5.

Yang, Shengxiang. (2008). Genetic algorithms with memory- and elitism-based immigrants in dynamic environments. *Evolutionary Computation*, v. 16, n. 3, p. 385–416. doi: 10.1162/evco.2008.16.3.385.

Zitzler, Eckart; Thiele, Lothar; Laumanns, Marco; Fonseca, Carlos M e Da Fonseca, Viviane Grunert. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation*, v. 7, n. 2, p. 117–132.