STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Statistics Opt. Inform. Comput., Vol. **X**, M 2026, pp 0–31.
Published online in International Academic Press (www.IAPress.org)

DPACO: Dynamic Tuning of ACO with Adaptive Strategy for QoS-Aware Web Service Composition

Naoufal El Allali [1,*], Abderrahim Zannou [2], Omayma Mahmoudi [1], Hakima Asaidi [1], Bellouki Mohamed [1]

[1] *MASI Laboratory, Multidisciplinary Faculty of Nador, Mohammed First University, Nador, Morocco.*
[2] *ERCI2A Laboratory, Faculty of Sciences and Techniques of Al Hoceima, Abdelmalek Essaadi University, Tetouan, Morocco.*

Abstract With the proliferation of services available on the Web, modern systems increasingly require mechanisms able to orchestrate multiple services in order to meet complex business requirements. Web service composition is an efficient approach to transform these basic services into coherent composite solutions, while ensuring the scalability and flexibility essential in distributed environments. However, Web service composition based on Quality of Service (QoS-aware WSC) presents a major problem. For each task in a workflow, several candidate services are available, characterized by significant heterogeneity and variability. In order to select the optimal combination of services able to satisfy QoS constraints transforms the problem into a complex multi-objective optimization challenge that is difficult to solve in polynomial time, especially in large-scale and dynamic environment. In this paper, we proposed a new approach called Dynamic Parameter Ant Colony Optimization (DPACO), an enhanced variant of the ACO algorithm for QoS-aware Web service composition. DPACO introduces adaptive parameter control and a dynamic pheromone injection strategy, which improve the balance between exploration and exploitation. This adaptivity helps to avoid stagnation and local optimal, while reducing search time and ensuring robustness in dynamic and heterogeneous environments. Experimental evaluations conducted on multiple datasets demonstrate the superiority of the proposed approach compared to classical ACO and its variants (SACO, MOACO, FACO, and EFACO). Specifically, it achieves improvements of up to +15.2% over ACO, +12.6% over SACO, +9.8% over MOACO, +7.3% over FACO, and +11.6% over EFACO in terms of solution quality. Moreover, the adaptive injection and parameter adjustment mechanisms significantly reduce the computational time required to reach optimal composite services.

Keywords Ant Colony Optimization (ACO), Adaptive Parameter Control, Combinatorial Optimization, Dynamic Parameter Ant Colony Optimization (DPACO), Dynamic Pheromone Injection Strategy, Meta-heuristic Algorithms, QoS-aware Web Services Composition (QoS-aware WSC)

AMS 2010 subject classifications 97Pxx, 97R50

DOI: 10.19139/soic-2310-5070-3355

1. Introduction

With the rapid integration of Artificial Intelligence (AI), the Internet of Things (IoT), and Cloud Computing into modern enterprise systems, Service-Oriented Architecture (SOA) [1, 2] has emerged as a fundamental framework enabling seamless interoperability between these technologies. It relies on Web Services, which act as a low-coupling, autonomous, interoperable, and platform-independent components, facilitating integration and communication between heterogeneous systems [3]. With a standardized ability to expose, discover, and compose functionality on a large scale, Web Services enable distributed systems to be flexible, scalable, and adaptive, and better address diverse business requirements in dynamic and widely distributed environments. Many companies such as Amazon,

*Correspondence to: Naoufal EL ALLALI (Email: n.elallali@ump.ac.ma). Department of Computer Science, Multidisciplinary Faculty of Nador, Morocco.

IBM, eBay, and Wachovia Bank have adopted this approach as the foundation of their distributed infrastructure. According to the latest reports on the SOA market, renowned vendors such as Oracle, Microsoft, AWS, and Google Cloud are cited as the main actors supporting this evolution [4].

In distributed environments, a single Web Service is usually not enough to meet diverse business needs, due to the increasing specialization of services and the heterogeneity of providers [5]. Web Service Composition (WSC) is therefore an essential approach, allowing several atomic services to be combined to create coherent composite services capable of performing complex business processes [6, 7, 8]. This mechanism enhances flexibility, modularity, and reusability, while allowing dynamic adaptation to variable execution contexts [9]. This issue has attracted growing attention within the scientific community due to its potential to enhance the agility, interoperability, and adaptability of distributed systems. Furthermore, it has become a major strategic issue for the development of service-oriented systems, in which the automation of the selection and orchestration process is crucial in dynamic and uncertain contexts [10, 11, 12].

With the rapid growth in the number of Web service providers, there are a massive number of services with similar features available at present. This selection process becomes even more complex when services must be selected based on quality of service (QoS) constraints, such as response time, cost, reliability, or availability. This problem, known as Quality of Service-aware WSC (QoS-aware WSC), consists to select the best combination of atomic services while respecting the overall QoS requirements of the system [13, 14]. It is widely recognized as a combinatorial problem [15] that cannot be solved efficiently in polynomial time for instances of realistic scale. In dynamic and uncertain environments, the task poses a significant optimization challenge due to the large number of possible solutions and the interdependence between the selected services.

Web service composition is a complex combinatorial optimization problem where each task is associated with a set of services that provide the same functionality but with different performance characteristics [16]. The selection of a service for each task generates an exponentially large solution space, which makes this a NP-hard problem. The complexity increases with the existence of multiple objectives, such as cost or response time, which must be minimized, and availability or reliability, which must be maximized [17]. Since these objectives are often conflicting, the algorithm must constantly balance them to arrive at an optimal composite solution. Furthermore, the dependencies between tasks can restrict the possible alternatives, which requires a global evaluation of each candidate solution rather than local decisions. In addition, the dynamic nature of service environments, where performance values may vary over time, also introduces additional uncertainty that must be considered during the optimization process.

Given the combinatorial and multi-objective nature of the QoS-aware WSC problem [18], traditional exact or exhaustive methods have become impractical, especially in dynamic and large-scale environments [19]. As a result, meta-heuristic approaches [20, 21, 22, 23] have emerged as effective alternatives, offering flexible strategies for exploring the vast solution space and approximating high-quality solutions within reasonable computation times. Among these, Ant Colony Optimization (ACO) algorithms [25, 26] have attracted particular interest due to their ability to achieve a balance between exploration and exploitation, adaptively search to find the best service combinations, and simultaneously deal with multiple QoS criteria. These properties make ACO and its variants particularly well suited to addressing the challenges inherent in-service composition that takes service quality into account.

Despite the effectiveness of variants of the ACO algorithm [27, 28, 29], they have notable limitations when applied to service composition that takes quality of service (QoS) into account. The balance between exploration and exploitation is often strained, which can lead to premature convergence toward sub-optimal solutions or, conversely, excessive exploration that slows convergence. Furthermore, static parameters, such as pheromone evaporation rates, the impact of heuristic information, and pheromone deposit quantities, reduce the adaptability of the algorithm to dynamic and heterogeneous environments [30, 31]. As a result, ACO may struggle to explore large search spaces effectively and respond to fluctuating service availability and varying QoS requirements. These challenges

motivate the development of improved and adaptive ACO variants that can dynamically adjust parameters and guide pheromone deposition more intelligently.

Since both the standard ACO and its variants have significant limitations, the EFACO (Enhanced Flying Ant Colony Optimization) algorithm [32] has been proposed to improve the quality of solutions while addressing some of the challenges inherent in-service composition. EFACO improves on the concept behind FACO (Flying Ant Colony Optimization) [33] by controlling the "flying" behavior of ants and depositing pheromones on neighboring services, which are selected at random. This enhancement allows for more flexible exploration of the search space, reducing the potential for premature convergence and better balancing exploration and exploitation. However, although EFACO improves the quality of the solution, it still has to deal with computational overhead due to the management of multiple pheromone trails and the stochastic process of neighbor selection, thus highlighting the need for additional adaptive strategies capable of effectively dealing with dynamic environments and QoS constraints.

These limitations demonstrate how important we need more intelligent and adaptive strategies that can maintain an efficient trade-off between search and discovery in dynamic service composition scenarios. With adaptive parameter tuning and targeted pheromone distribution, the search can focus on more promising areas of the solution space, to quickly achieve high-resolution composite services, with a high level of robustness and stability. Such approaches are essential to deal with the combinatorial complexity and multi-objective nature of QoS-aware service composition, especially in environments characterized by variable service availability and provider heterogeneity.

In this paper, we propose the DPACO (Dynamic Parameter Ant Colony Optimization) approach as a cross-objective improvement over EFACO, taking advantage of the multi-objective concepts used in EFACO while introducing adaptive strategies to overcome its limitations. More specifically, DPACO dynamically adjusts pheromone deposition and tweaks algorithm parameters to ensure a more efficient balance between exploration and exploitation. This allows the algorithm to navigate efficiently in large and heterogeneous service spaces, respond to dynamic variations in service availability, and guide the search toward solutions that satisfy various quality-related objectives. By incorporating these adaptive mechanisms, DPACO improves convergence speed, solution quality, and stability, while resolving the issues of computational overload and stochastic neighbor selection observed in EFACO.

To further validate the effectiveness of the proposed approach, extensive experiments were conducted, comparing DPACO with several ACO variants across diverse scenarios. The results demonstrate that the adaptive strategies not only improve convergence speed but also enhance the stability and reliability of the solutions. By combining dynamic parameter adjustment with intelligent pheromone deposition, the approach efficiently identifies an optimal composition of services while effectively managing the trade-off between exploration and exploitation. These findings underline the capacity of the method to handle large-scale, heterogeneous, and dynamic service environments, ensuring high-quality QoS-aware compositions in complex workflows.

The remainder of this paper is organized as follows. Section 2 defines the QoS-aware web service composition problem and presents the baseline ACO algorithm along with its enhanced variant EFACO, highlighting their principles and limitations. Section 3 provides a review of related works, discussing previous approaches and their respective advantages and shortcomings. Section 4 presents the proposed DFACO approach, detailing its adaptive mechanisms for dynamic parameter tuning and intelligent pheromone deposition. Section 5 reports the experimental evaluation and discusses the results obtained across various scenarios, including comparisons with existing ACO variants. Finally, Section 6 concludes the paper, summarizing the main contributions.

2. Preliminaries

2.1. Problem Description

This section covers the mathematical modeling of the Web Service Composition problem and the service composition workflow, before we propose our approach. WSC problem formulated as a combinatorial problem, where the objective is to select, for each task T_i, a service S_k from a set of service candidates S^* are available, offering the same functionality, while optimizing multiple quality of service (QoS) criteria such as cost, response time,

availability, and reliability (As illustrated in Table 1). Due to the exponential growth of the search space, finding an optimal service composition becomes a complex challenge.

As illustrated in Fig. 1, let's consider a workflow W composed of a set of tasks $T = \{T_1, \ldots, T_n\}$, where each task T_i is executed by selecting one of the available workflow services (WS). Let $S_i = \{WS_1, \ldots, WS_m\}$ be the set of m services available for task T_i. The aim is to select one service per task so as to minimize a cost or maximize a quality of service (QoS) while respecting workflow constraints. In this work, we consider four QoS features, which represent non-functional constraints on services:

- **Cost (S):** Denotes the total cost of the services used in the solution S.
- **Response time RT(S):** Defines the total workflow execution time.
- **Availability A(S):** Indicates the probability of successful invocation of selected services.
- **Reliability R(S):** Represents the message transmission error rate.

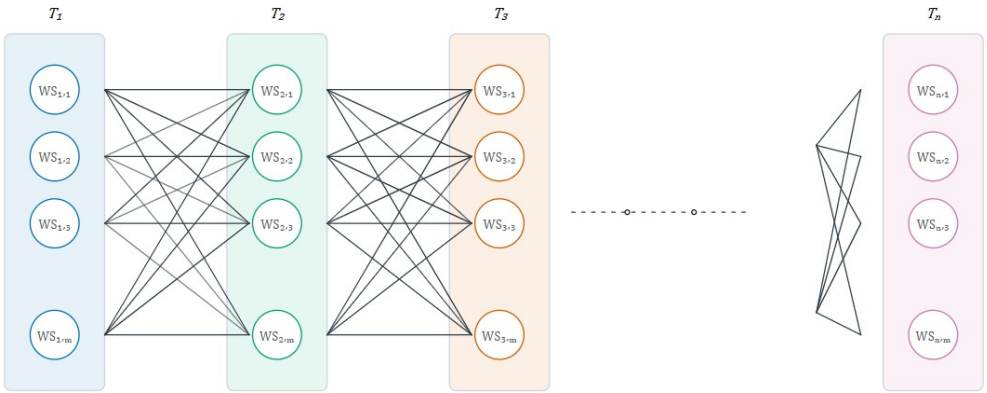


Figure 1. The model for WSC assigned to each task

QoS Criteria	**Expression**
Cost (C)	$\sum_{i=1}^{n} c(ws_{i,j})$
Response Time (RT)	$\sum_{i=1}^{n} RT(ws_{i,j})$
Throughput (T)	$\prod_{i=1}^{n} T(ws_{i,j})$
Reliability (R)	$\prod_{i=1}^{n} R(ws_{i,j})$

Table 1. Aggregation Models for QoS Value Computation

The quality of the solution S is evaluated in terms of the values of the characteristics C(S), RT(S), A(S) and D(S), as defined in Table 1. These criteria allow us to compare different solutions and identify those that best balance the constraints. Thus, the problem is formalized as a multi-objective combinatorial optimization, where the objective is to minimize C(S) and RT(S), while maximizing A(S) and D(S). This makes service selection a complex problem,

requiring advanced approaches to identify solutions to satisfy these constraints, while ensuring an optimum balance between these different functions.

Workflows are classified as simple or complex depending on their structure. A simple workflow follows sequential execution only, while a more complex workflow combines multiple execution modes. Fig. 2 illustrates four basic models:

- **Sequential (a):** This model is based on a linear execution where each task T_i is followed directly by T_{i+1}. It is represented as:
$$T_1 \to T_2 \to T_3 \to \cdots \to T_{n-1} \to T_n$$
- **Parallel (b):** Tasks can be executed simultaneously, reducing total execution time.
$$T_1 \to (T_2 \parallel T_3) \to T_4$$
- **Conditional (c):** This model introduces a bifurcation where a choice is made between several paths depending on certain conditions. A conditional workflow can be formalized as:
$$T_1 \to \begin{cases} T_2 & \text{if condition } P1 \\ T_3 & \text{if condition } P2 \end{cases} \to T_4$$
where $P1$ and $P2$ are probabilities associated with possible paths.
- **Loop (d):** There are workflows that require a task to be repeated before the next step can be taken. A loop is represented mathematically as:
$$T_1 \to T_2 \to \cdots \to T_k \to T_1$$

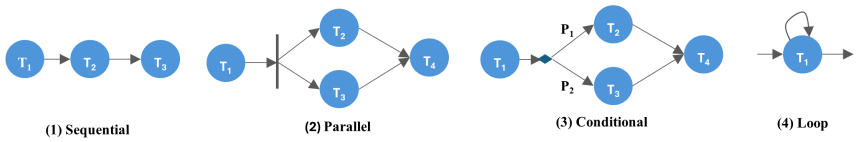


Figure 2. The workflow models

These heterogeneous models introduce a combinatorial complexity that makes their execution and optimization notably complex. As illustrated in the Fig. 3, To deal with these structures consists the decomposition a complex workflow into independent sub-workflows and the optimization of each subset in a targeted manner.

Fig. 3 (a) represents a complex workflow, which combines several execution patterns and has dependencies that increase the search space for optimal solutions. Fig. 3 (b) illustrates its decomposition into simpler workflows, each considered as an individual problem. This segmentation reduces the dimensionality of the problem, limiting the propagation of interdependencies and facilitating the evaluation of optimal solutions in a reduced search space. Since these simple workflows are represented as autonomous entities, their further decomposition is neglected. The optimal solutions are then selected by comparing the fitness values of the sub-workflows, according to the defined multi-objective constraints and objectives.

The main challenge of WSC is to find the optimal combination of Web services (WS) to achieve the best compromise between QoS criteria. Each solution corresponds to a complete composition path, where a specific service is selected for each task. Since the number of potential compositions increases exponentially with the number of tasks, the total number of potential solutions for this problem corresponds to: m^n.

Given that the number of possible service compositions grows exponentially as m^n, the problem becomes computationally intractable for exhaustive search approaches. This complexity classifies WSC as an NP-hard problem, requiring efficient heuristic or metaheuristic approaches to explore the vast solution space and identify near-optimal or optimal compositions.

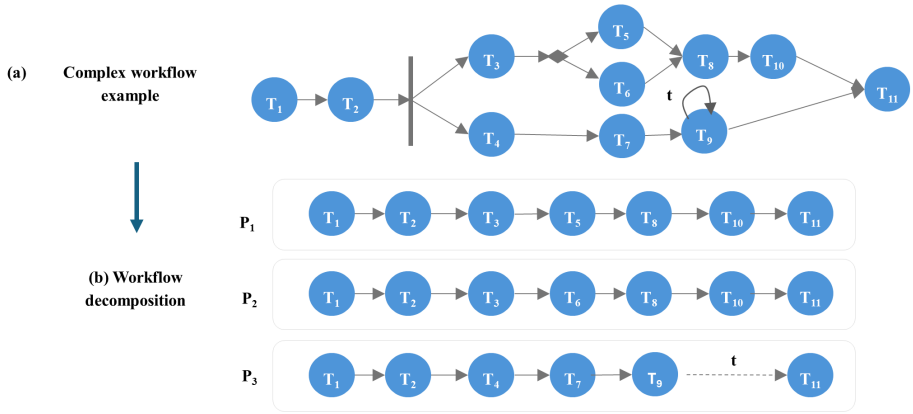


Figure 3. Decomposition of a complex workflow into independent sub-workflows

2.2. ACO

The ACO algorithm has recently demonstrated how it can be used as an optimization metaheuristic for many NP-hard problems, such as the travelling salesman problem or job shop scheduling . Moreover, it has been widely explored in the context of dynamic WSC, a context with characteristics similar to those of large combinatorial optimization problems, notably due to the exponential size of the search space and the multiple constraints related to QoS.

In the context of Web service composition, each solution represents a sequence of services $C = \{WS_1, \ldots, WS_n\}$, constructed progressively by a population of artificial ants. At each stage, an ant chooses a service WS_k from a set of candidates S_i to respond to a task T_i, according to a probability defined by the formula:

$$P_{i,j}^{k}(t+1) = \frac{[\tau_{i,j}]^{\alpha}[\eta_{i,j}]^{\beta}}{\sum_{WS_i, l \in S_i}[\tau_{i,l}]^{\alpha}[\eta_{i,l}]^{\beta}} \ \epsilon \ N_i^k \tag{1}$$

Where α and β represent control parameters that respectively determine the importance assigned to pheromone intensity $\tau_{i,j}$ and heuristic value $\eta_{i,j}$ when selecting a service. The set N_i^k denotes the services still available to ant k at step i, i.e., the set of compatible candidates for functional task T_i that have not yet been selected in the current composition construction.

In the ACO algorithm applied to service composition, local updating allows each ant to modify the pheromone trace, after a service has been selected for a task, in order to reduce its attractiveness temporarily and encourage the exploration of alternatives. This operation is generally defined by the formula:

$$\tau_{i,j}(t+1) = (1-\rho)\,\tau_{i,j}(t) + \rho\,\tau_0 \tag{2}$$

where ρ is the local evaporation rate, and τ_0 the initial amount of pheromone. After all the ants have successfully constructed their solutions, a global pheromone update is applied to reinforce the service choices involved in the best compositions found. This process aims to guide the search towards the most promising solutions, thus reinforcing their attractiveness in the following iterations. The global update is typically defined by:

$$\tau_{i,j}(t+1) = (1-\rho)\,\tau_{i,j}(t) + \rho\,\Delta\tau_{i,j} \tag{3}$$

where $\Delta\tau_{i,j}$ represents the pheromone augmentation provided by the best solutions, often calculated as:

$$\Delta\tau_{i,j}^{(k)} = \begin{cases} \dfrac{1}{f(c^*)}, & \text{if } (i,j) \in c^* \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where c^* represents the best composition found and $f(c^*)$ its objective value.

The ACO algorithm has been proved a highly efficient method to address a wide range of combinatorial optimization problems, notably in terms of its ability to efficiently explore large search spaces and converge on high-quality solutions. Nevertheless, it presents some limitations. In particular, colonies may converge to sub-search spaces prematurely, which leads to stagnation and loss of diversity. The balance between exploration and exploitation depends highly on the parameters α, β, and ρ, whose manual adjustment limits the robustness of the algorithm to dynamic environments. Moreover, as the size of the search space increases, convergence tends to slow significantly, and the absence of adaptive mechanisms reduces its ability to respond to changing contexts.

The EFACO algorithm [32] has proved to be an efficient approach to overcome a few of the limitations of ACO, in particular the problem of stagnation in a local optimum. However, this approach presents a non-negligible computational overhead, as the ants must, at each iteration, randomly explore the set of neighboring services on the solution path. In contrast, our proposed approach, detailed in Section 4, adopts a more intelligent neighbor selection strategy, in order to better balance exploration and exploitation. In addition, it aims to reduce the sensitivity of the algorithm to initial parameter settings.

2.3. EFACO

The EFACO algorithm is an enhancement of the ACO algorithm, specifically the FACO, designed to improve the control of the balance between exploration and exploitation in the search space. It has been designed as a more efficient neighbor selection process and a multi-pheromone distribution in order to improve the quality of solutions, maintain diversity, avoid stagnation and minimize computational overload. it has also been generalized to address the problem of WSC.

In the EFACO algorithm, the transition process allows each ant to select the next service based on two factors: the value of the pheromones and a local heuristic related to the QoS criteria. While FACO relies on a single trace, EFACO uses a multi-pheromone mechanism where the pheromone is represented by a set of values associated with different QoS attributes. The probability that an ant k will choose a service s' for task $t+1$ is given by:

$$P^k_{(t,s)(t+1,l)} = \frac{[\tau_{(t,s)(t+1,s')}]^\alpha [\eta_{(t,s)(t+1,s')}]^\beta}{\sum_{l \in N^k_{t+1}} [\tau_{(t,s)(t+1,l)}]^\alpha [\eta_{(t,s)(t+1,l)}]^\beta} \quad \in N^k_t \tag{5}$$

The pheromone associated with a transition is calculated as the sum of the traces of each attribute:

$$\tau_{(t,s)(t+1,s')} = \tau^A_{(t,s)(t+1,s')} + \tau^R_{(t,s)(t+1,s')} + \tau^C_{(t,s)(t+1,s')} + \tau^{RT}_{(t,s)(t+1,s')} \tag{6}$$

The local heuristic is defined by a combination of the various QoS dimensions:

$$\eta_{(t,s)(t+1,s')} = \eta^A_{(t,s)(t+1,s')} + \eta^R_{(t,s)(t+1,s')} - \left(\eta^C_{(t,s)(t+1,s')} + \eta^{RT}_{(t,s)(t+1,s')}\right) \tag{7}$$

The quality of the solutions is then evaluated by a function that aggregates these criteria according to weights defined by the preferences of the user:

$$\arg\max_{k \in \text{ants}} f = \frac{W_a \prod_{i=1}^{n-1} A^k_{(t,s)(t+1,s')} + W_r \prod_{i=1}^{n-1} R^k_{(t,s)(t+1,s')}}{W_c \prod_{i=1}^{n-1} C^k_{(t,s)(t+1,s')} + W_{rt} \prod_{i=1}^{n-1} RT^k_{(t,s)(t+1,s')}} \tag{8}$$

where W_a, W_r, W_c, and W_{rt} are the weights associated with the QoS attributes according to user preferences, and n the number of tasks in the composition.

In the step followed, the ants update the pheromones locally on their path, using the same classic formula as in ACO. They deposit traces on their path to encourage other ants to follow these routes in future iterations.

$$\tau_{(t,s)(t+1,s')} = (1-\rho)\,\tau_{(t,s)(t+1,s')} + \rho\,\tau_0 \tag{9}$$

where $(t,s)(t+1,s')$ represents the amount of pheromone deposited on the edge connecting a task service t to a task service $t+1$. The value τ_0 denotes the initial pheromone, while ρ corresponds to the evaporation rate.

After the solutions have been constructed and local updates have been applied by each ant, EFACO applies a global pheromone update mechanism to reinforce the transitions used by the best solution that has been found. This process enhances convergence towards high-quality paths, while controlling the overstating of certain solutions by evaporating traces on other edges. This is realized using the following formulas for global pheromone updates:

$$\tau_{(t,s)(t+1,s')} = (1-\rho)\,\tau_{(t,s)(t+1,s')} + \rho\,\Delta_{(t,s)(t+1,s')} \tag{10}$$

where $\rho \in (0,1)$ is the pheromone evaporation factor. The amount of pheromone released is calculated as follows:

$$\Delta\tau_{i,j}^{(k)} = \begin{cases} \dfrac{1}{l_{\text{best}}}, & \text{if the edge belongs to the best solution} \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

where l_{best} denotes the length of the best global solutions. After the global update of pheromones, the EFACO algorithm adds a step to select neighbors, which is a major improvement to the FACO flight process. Unlike the latter, which exhaustively explores all neighbors, EFACO applies an adaptive selection method that aims to reduce computational overload while maintaining a good balance between exploration and exploitation. At each iteration, a random set of S_N neighboring services is selected, then sorted according to their Euclidean distance to a service $WS_{i,x}$ belonging to the best solution (as illustrated in Fig. 4). The distance between two services is defined by the following formula, taking into account QoS attributes (cost, response time, availability, reliability):

$$d_{(s,y)} = \sqrt{(C_s - C_y)^2 + (RT_s - RT_y)^2 + (A_s - A_y)^2 + (R_s - R_y)^2} \tag{12}$$

The pheromone injection mechanism is the next step, applied to neighboring services, favoring those that are most promising based on their proximity and heuristic values. The amount of pheromone deposited on neighbors $l \in N_s$ is adjusted according to:

$$\tau_{(t+1,s')(t+1,l)} = \tau_{(t+1,s')(t,l)} + \left(\frac{\tau_{(t+1,s')(t,s')}}{1 + \sqrt{d'\left(\eta_{(t+1,s')(t+1,l)}\right)}} \right) \tag{13}$$

where $d'\left(\eta_{(t+1,s')(t+1,l)}\right)$ denotes the normalization of the distance between a Web Service belonging to task $t+1$ and a neighbor Web Service l of the same task, with $l \in N_s$. This normalized distance is calculated using the following equation:

$$d'\left(\eta_{(t+1,s')(t+1,l)}\right) = \frac{\eta_{(t+1,s')(t+1,l)}}{\sum_{q=1}^{NS} \eta_{(t+1,s')(t+1,q)}} \tag{14}$$

Finally, the number of neighbors considered is dynamically adjusted according to:

$$y = \left(\frac{R}{j}\right) \times S_N \tag{15}$$

where $R \in [0,1]$ is a random variable and j is the current iteration number. This mechanism allows us to increase exploration during the first iterations and to encourage exploitation in the final iterations.

Recall that the EFACO algorithm was introduced to address the computational cost and stagnation problems in FACO by integrating multi-pheromone mechanisms and adaptive neighbor selection. However, certain limitations persist, especially in handling dynamic environments and maintaining an optimal balance between exploration and exploitation. In this work, we propose additional improvements to address these weaknesses and enhance the overall performance of the EFACO algorithm.

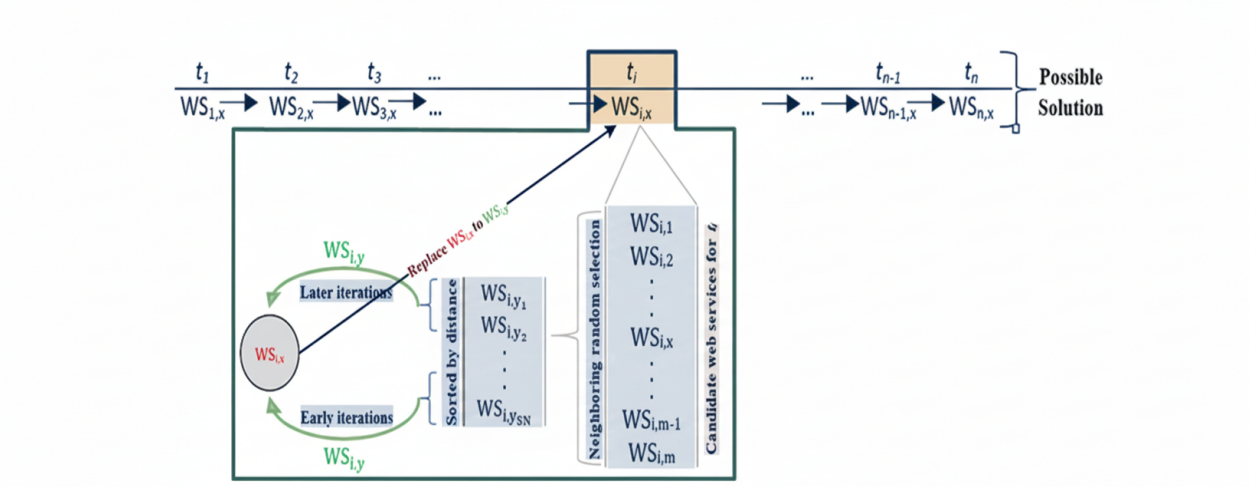


Figure 4. Neighboring selection Process [32]

3. Related Works

As distributed systems have become more complex, particularly with the rapid growth of the Internet of Things (IoT), the WSC based on QoS has become one of the major challenges in combinatorial optimization. Due to its high complexity (NP-hard), much research has focused on swarm intelligence algorithms, such as ACO, PSO, and Genetic Algorithms (GA), to address this challenge efficiently.

Qiqing et al [34] proposed an algorithm called MOACO for Web service selection, aiming at multi-criteria QoS optimization with a Pareto optimal set and adaptive pheromone adjustment to achieve a better trade-off between exploration and exploitation. Li et al [35] proposed a multi-objective chaotic ant colony optimization (MOCACO) system, incorporating a chaos operator to accelerate convergence speed and improve search performance. Xia et al. [36] proposed a multi-pheromone strategy in standard ACO to improve the search for near-optimal solutions on a composition graph. Wang et al. [37] proposed an adaptive algorithm ACO, to dynamically adapt pheromone evaporation according to the degree of confidence, in order to enhance the robustness and diversity of solutions generated in dynamic environments. Chifu et al [38] have developed an ACO-based composition selection method using a 1-OPT heuristic, expanding the search space to avoid stagnation on local solutions. WU and Zhu [39] improved on ACO by modeling composition as an acyclic directed graph (DAG) and integrating a multi-criteria evaluation function to simultaneously optimize QoS and transactional constraints, thereby enhancing solution quality and limiting stagnation. Yu et al. [40] proposed ACO-WSC, the adaptation of ACO for multi-cloud service composition with a directed graph and hybrid evaluation function, to improve search efficiency and to accelerate convergence. Dahan et al. [41] extended ACO with DFACO, adding dynamic neighbor selection and 3-Opt to balance exploration/exploitation and limit stagnation. AlAyed et al. [28] enhanced ACO with a swap strategy and multi-pheromone to diversify search and accelerate convergence. Dahan et al [32] refined the model by proposing EFACO, combining optimized neighbor selection and multi-pheromone distribution to improve the efficiency and quality of solutions.

Several approaches have resorted to combine it with other optimization methods or integrate it into multi-agent systems, to enhance exploration, provide convergence stability and meet the challenges of distributed environments. Zhao et al [42] developed a hybrid algorithm where ACO generates initial non-dominated solutions, then improved by GA, in order to achieve a balance between exploration and exploitation and to speed up the achievement of optimal solutions. Dahan et al [43] proposed an efficient multi-agent ACO algorithm for cloud service composition, to improve the exploration/operation balance via dynamic agent distribution and adaptive update mechanisms. Meanwhile, El Allali et al. [22] introduced a framework for composition in heterogeneous environments by

combining ACO with mobile agents, enabling intelligent reduction of the search space and improved scalability and composition execution time. Gohain and Paul [44] combined PSO and ACO for Web service composition, taking advantage of PSO's exploratory capability and ACO's local exploitation to improve the overall quality of compositions. Du and Miao [45] introduced Be-ACO, merging the Beetle Antenna Search (BAS) algorithm with ACO, to dynamically generate search subspaces and improve solution accuracy while avoiding local optimal. Yang et al. [46] and Bei et al. [47] explored hybrid approaches to improve service composition optimization in multi-cloud environments. Yang et al. developed a dynamic hybrid algorithm, DAAGA, combining ACO and GA, with an adaptive mechanism based on an iterative adjustment threshold to balance global and local search, thus accelerating convergence and improving the stability of large-scale solutions. For their part, Bei et al. proposed an improved ACO incorporating a multi-pheromone mechanism to specifically optimize different QoS parameters, and introduced a genetic (GA) mutation operation to avoid stagnation and enhance solution diversity. Qi et al [48] combined the Skyline algorithm with ACO for large-scale transactional service selection, using Skyline to reduce the search space and ACO to optimize QoS and transactional constraints, thus to improve the efficiency and quality of the solutions generated.

Authors	**Methodology**	**Pros**	**Cons**
[36] Xia et al (2008)	Enhanced ACO with multi-pheromones to dynamically optimize web service composition based on variations in QoS and service state.	- Optimizes multiple QoS simultaneously. - Adapts to real-time service changes. - Medium stagnation reduction. - Improved convergence speed.	- Sensitivity to initial parameter settings. - High computational complexity due to multi-pheromone management. - Premature convergence in dynamic environments.
[34] Liu Qing et al (2009)	Multi-objective ant colony optimization (MOACO) based on Pareto dominance for dynamic Web service composition under global QoS constraints.	- Multi-criteria optimization without weighting. - Diversity of Pareto solutions. - Pheromone reset to minimize stagnation. - Reduced execution time compared to MOGA.	- Low convergence without local optimum. - Sensitivity to parameters (α, β, ρ). - Not adapted to complex workflows. - Residual risk of stagnation.
[35] Li et al (2010)	Improved MOACO using chaotic updating (Logistic Map) to enhance search diversity and avoid premature convergence.	- Reduced convergent risk to local optimal. - More Pareto optimal solutions. - Low execution time.	- Sensitivity to chaotic parameter (μ). - Higher complexity. - No dynamic adjustment of chaotic parameters.
[39] Wu et al (2013)	ACO applied to Web service composition taking into account both QoS constraints and transactional properties, as modeled by an acyclic directed graph.	- Simultaneous optimization of QoS and transactional constraints. - Generic model capable of handling complex workflows. - Reduces failures due to transactional inconsistency.	- High complexity due to overlapping QoS and transactional constraints. - Sensitivity to parameter settings of ACO factors (pheromones, weights). - Risk of stagnation without dynamic parameter adjustment.

Table 2. Overview of ACO algorithm improvements for QoS-aware composition

[38] Chifu et al (2014)	ACO is adapted with the addition of a 1-OPT heuristic to avoid local stagnation, using an enhanced planning graph (EPG) integrating semantic similarity and QoS.	- Integration of QoS and semantic similarity criteria. - Avoid stagnation with 1-OPT heuristic. - Reasonable convergence time. - Adaptability to complex service compositions.	- Sensitive to initialization parameters. - Risk of sub-optimal solutions in large search areas. - Using the EPG leads to a higher cost in terms of complexity.
[37] Wang et al (2014)	Adaptive Ant Colony Optimization with dynamic adjustment of the pheromone evaporation rate, integrating QoS and trust degree for dynamic Web service selection.	- Integration of QoS and trust criteria. - Dynamic parameter adjustment. - Reduced stagnation. - High convergence and increased Pareto solutions.	- Sensitivity to adaptive parameter settings. - High computational complexity. - Poor tuning may degrade performance.
[42] Zhao et al (2014)	Hybrid algorithm using improved ACO to generate non-dominated solutions and GA to narrow the search and optimize the selection of QoS-constrained Web services.	- High-quality multi-objective solutions. - Combination of the advantages of ACO (exploration) and GA (exploitation). - Higher quality results on large datasets.	- High complexity due to ACO-GA alternation. - Sensitivity to parameters (pheromone intensity, mutation rate, crossover). - High memory consumption with hybrid population control. - Difficult to balance exploration and exploitation phases without adaptive adjustment. - Risk of increasing computing time on very large sets.
[40] Yu et al (2015)	ACO used for the composition of web services on multi-cloud environments, aims to minimize the number of clouds used in the composition.	- High performance in multicloud composition. - Low-cost communication between clouds. - Ability to quickly find valid compositions. - Robust approach to combinatorial complexity.	- Sensitivity to ACO parameters (pheromones, heuristics). - Failure to dynamically adapt parameters to balance exploration and exploitation. - Complexity increases with the number of services and clouds.
[44] Gohain et al (2016)	Hybrid algorithm using Particle Swarm Optimization (PSO) for fast initialization and Ant Colony Optimization (ACO) to improve convergence in QoS-aware web service composition.	- Rapid exploration with PSO. - Smooth optimization with ACO. - High-quality solutions compared with PSO alone.	- High hybridisation complexity. - Sensitivity to PSO and ACO parameters. - Higher computational cost. - Difficulty to balance the impact of PSO and ACO. - Precise tuning required for good performance.

Table 2. Overview of ACO algorithm improvements for QoS-aware composition (continued)

[28] AlAyed et al (2019)	SACO combine adaptive ant flight, optimized neighbor selection, and multipheromones for QoS-aware composition.	- Improved dynamic adaptation for a better balance between exploration and exploitation. - Reduced execution time through flight limitation. - High robustness on different types of workflows.	- High number of iterations required to reach a satisfactory solution, especially without dynamic adaptation. - Sensitivity to adaptive settings. - Performance dependent on threshold calibration. - High theoretical computation time growth with problem size (e.g. $O(n^3)$, $O(n^4)$).
[41] Dahan et al (2019)	Improved FACO algorithm with dynamic neighbor selection, integration of 3-Opt, and adjustment of the number of neighbors injected depending on the quality of the solution to solve the TSP.	- Improved balance between exploration and exploitation through dynamic pheromone injection. - Low execution time.	- More complex algorithm due to dynamic neighbor management. - Sensitivity to threshold settings for dynamic adjustment. - Specific adjustments may be required depending on the size of the problem.
[46] Yang et al (2019)	Hybrid algorithm using ACO (Max-Min) for global search and GA for local optimization, with dynamic merge threshold adjustment for QoS-aware composition.	- Rapid convergence through dynamic adaptation. - Reduced risk of stagnation. - Suitable for large-scale problems.	- High memory requirements. - Sensitivity to dynamic settings. - High computational overhead on very large datasets.
[32] Dahan et al (2021)	Enhanced FACO (EFACO) integrating restricted ant flight, adaptive neighbor selection, and multi-pheromone distribution for QoS-aware composition.	- Enhanced QoS optimization via multi-pheromones. - Improved balance between exploration and exploitation through adaptive selection. - Significant reduction in execution time. - Fast convergence and high solution quality.	- High complexity due to the handling of several pheromones. - Sensitivity to adaptive threshold settings. - High memory cost to buffer multiple pheromone matrices. - Possible reduction in solution diversity if parameters are poorly adjusted. - Difficulty of optimal parameterization in highly dynamic environments.

Table 2. Overview of ACO algorithm improvements for QoS-aware composition (continued)

[43] Dahan et al (2021)	Distributed ACO using a multi-agent system (MAACS) combining several agents for execution and monitoring to balance the search for solutions in the composition of QoS-aware cloud services.	- Enhanced balance between exploration and exploitation. - Low probability of premature convergence. - High performance stability on large datasets. - Short run-time due to parallel execution.	- High memory cost due to multiple sub-populations. - Sensitivity to the number and type of agents. - Potential degradation if agent distribution is misconfigured. - Limited scalability for extreme workflow sizes.
[22] El Allali et al (2022)	An improved ACO built into a mobile multi-agent system for dynamic composition of web services, with sub-problem partitioning and parallel execution to maximize performance and scalability.	- Enhanced balance between exploration and exploitation. - Short run-time due to parallel execution. - Adaptable to heterogeneous environments.	- Complex multi-agent coordination. - Sensitivity to ACO parameters. - High computational cost. - Possible inefficiency if agents are poorly distributed.
[45] Du and Miao (2022)	Hybrid algorithm combining BAS for global exploration and ACO for local optimization in QoS-constrained web service composition.	- Improved convergence by combining the speed of BAS with the precision of ACO. - Reduced risk of local stagnation. - Dynamic adaptation to the search space.	- High complexity due to BAS-ACO integration. - Fine-tuning required to balance exploration and exploitation. - High computational costs in the case of large service sets.
[47] Bei et al (2024)	Enhanced ACO with multi-pheromone mechanism for QoS optimization, combined with genetic mutation to avoid stagnation and accelerate convergence in multi-cloud environments.	- Efficient optimization of multiple QoS. - Low mutation stagnation. - Fast, stable convergence. - Adaptable to multi-cloud architectures. - Improved latency and response time management.	- High complexity due to multipheromone management and mutation. - Sensitivity to pheromone weight settings. - High memory consumption. - High computational cost for massive instances. - Need for precise adjustments to balance multiple QoS.
[48] Qi et al (2024)	Using Skyline to reduce the search space and ACO to optimize the selection of Web services with QoS and transactional constraints.	- Efficient reduction of search space via Skyline. - Combined QoS and transactional optimization. - Reduced risk of local stagnation. - Improved speed and quality of solutions.	- High complexity of Skyline pretreatment. - Sensitivity to ACO parameters (pheromones, volatility). - Significant computational cost on very large datasets. - Possibility of memory overload with very large numbers of services. - Difficulty adjusting Skyline/ACO thresholds according to context.

Table 2. Overview of ACO algorithm improvements for QoS-aware composition (continued)

The contributions illustrated in the comparison table presented previously cover a large number of different extensions and enhancements to the Ant Colony Optimization (ACO) algorithm, applied to the dynamic composition of Web services in the context of Quality of Service (QoS) constraints. These approaches generally belong to three main categories:

- **Improved approaches**, aimed specifically to modify the internal inner core of the ACO algorithm (e.g. via multi-pheromone mechanisms, mutation strategies or adaptive update rules) in order to increase convergence, avoid local stagnation and achieve a better balance between exploration and exploitation;
- **hybrid approaches**, which combine ACO with other metaheuristics such as genetic algorithms (GA), PSO or BAS to combine the advantages of several optimization strategies;
- **distributed or multi-agent approaches**, based on a decentralized architecture to improve scalability, distribute the workload and adapt composition to complex cloud environments.

Although these different approaches have brought significant enhancements as regards solution quality, diversity and robustness, they share common limitations such as high complexity, sensitivity to parameter settings, high memory consumption, and difficulty in large-scale adaptation.

In order to address these limitations, the present study builds on the philosophy behind some advanced ACO algorithm versions, in particular those inspired by EFACO. The aim is to limit the stagnation problem, achieve faster convergence, and improve the adaptability of the algorithm through dynamic parameter adjustment and more efficient pheromone updating. With these enhancements, the algorithm is adopted to deal more efficiently with the constantly growing complexity of large-scale environments, where the large number of services and the dynamic nature of interactions require quick and relevant decisions.

In the following paragraphs, we will describe the main elements of the proposed approach and how it can overcome the limitations mentioned above.

4. Proposed Approach

EFACO is an extension of the FACO algorithm proposed by the paper [33], specifically designed to solve the QoS-aware WSC problem. This alternative algorithm is aimed to make up the gaps in FACO, and includes improvements such as ant flight limitation, random selection of a subset of neighbors to speed up the search, and a multi-pheromone mechanism to balance QoS criteria more efficiently. However, EFACO still presents some limitations related to the narrow update of pheromones, neither taking into account the holistic quality of explored solutions nor efficient redistribution to neighboring services, which in turn can lead to stagnation in sub-optimal solutions. Although EFACO outperforms FACO, further improvements are needed to enhance its robustness and ensure optimum convergence.

In order to improve solution quality, this section presents a new method called DPACO (Fig. 5 shows the DPACO flowchart), which combines the exploratory strengths of ACO with the convergence approaches of Grey Wolf Optimization (GWO) [49]. Through using proportional and dynamic pheromone updating as well as intelligent parameter adjustment over time, this approach solves EFACO's constraints and reduces stagnation risk, hence strengthening convergence. The main novelty of DPACO is its capacity to incorporate all QoS parameters while yet efficiently balancing exploration and exploitation. Moreover, its adaptive nature allows it to dynamically respond to unforeseen variations, ensuring robust and flexible optimization, particularly in heterogeneous and evolving environments. Even though this algorithm provides a solution to the WSC problem, it also addresses a wide range of optimization problems where efficient exploration and fast convergence are required.

Fig. 5 outlines the steps of the improved DPACO algorithm. This algorithm outperformed the traditional ACO and EFACO in terms of solution quality. Since the traditional ACO and EFACO algorithm has 3 main factors:

- **The first factor** is to exclude the exploitation of other solutions, as the high level of pheromone in the trails, which reduces the diversity and converges rather too quickly to a sub-optimal solution. **(Stagnation Problem and Diversity)**

- **The second factor** consists in the parameters in the algorithm, which means marginal control of the balance between exploitation and exploration, which in turn can affect the quality of the solution, especially in a changing environment. **(Balance between exploitation and exploration)**
- **The last factor** is the high cost of computing on a large problem, which is often quadratic or more in complexity when the number of solutions explored increases, especially when processing is done in real time. **(Computation time)**

Therefore, our idea addresses all three aspects since the composition problem expects a system to identify the optimal Web Service Composition (WSC) inside a reasonable time. Moreover, a harmony between exploration and exploitation has to be reached to grow efficiency and solution quality. Thus, the system has to reach an optimal composition when it is to satisfy the above-mentioned factors.

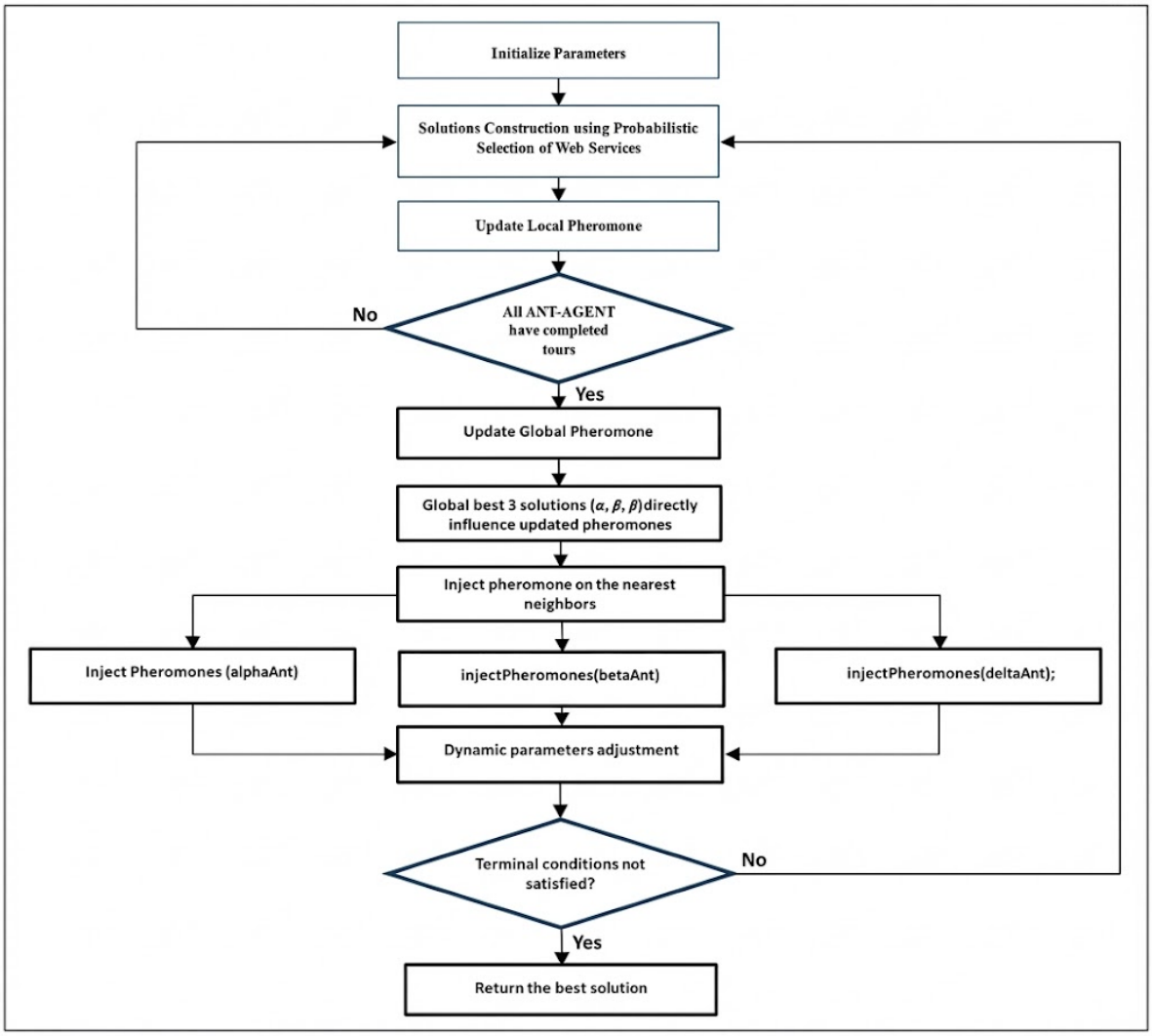


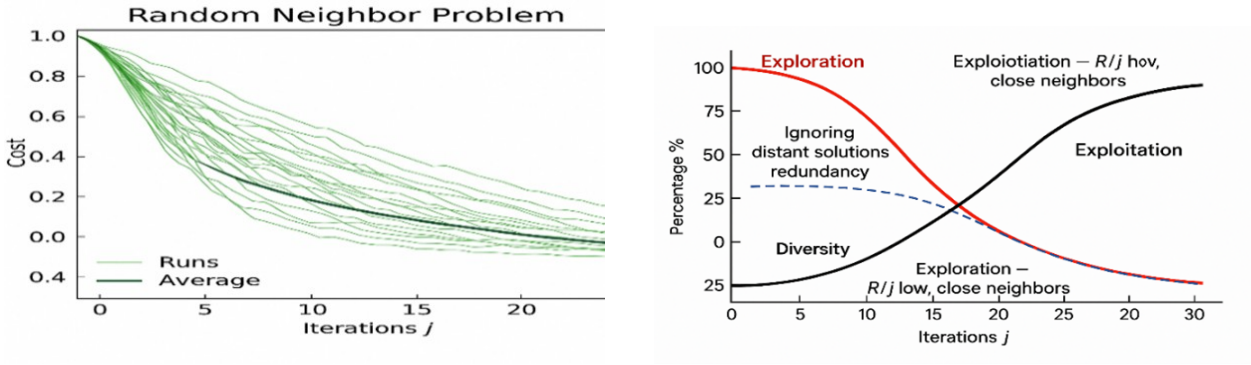
Figure 5. Flowchart of DPACO algorithm

4.1. Proportional Pheromone Distribution

In the EFACO algorithm, based on a random search of neighboring S_N services to enhance the injection of pheromones in other paths, these services are sorted according to their distance from a random Web service that belongs to the best solution found. The pheromone injection on these S_N random neighbor services must belong to the same task T_i, this algorithm uses the same formula as used in FACO (see the equation 14). However, the results obtained by the EFACO algorithm present a marginal domination over FACO to enhance their probability to be selected in the next iterations in order to harmonize loads between exploration and exploitation phases.

Despite the performance of EFACO, introduces an adaptive strategy between exploration and exploitation, but this flexibility raises a number of negative factors. Recourse to a random factor in the neighbor selection process can lead to inherent stochastic variability in each run, which can affect convergence stability and result reproducibility. In addition, the formula 15 used to dynamically adjust the neighbors induces a rapid reduction in exploration along the iterations. This rapid transition to exploitation limits the diversity of solutions generated, favoring premature convergence towards sub-optimal solutions, which increase the risk of stagnation in the search space and convergence towards sub-optimal solutions. Even if S_N is random, sorting neighbors by distance alone may discard more distant promising solutions, favoring redundancy at the expense of diversity.

Therefore, the EFACO algorithm could prematurely converge on sub-optimal solutions and stagnate in a locally satisfactory zone, without sufficient diversification in the advanced optimization phases. This is mainly due to the neighbor mechanism used during optimization. Indeed, at the beginning, when the number of iterations j is low, the ratio $\frac{R}{j}$ is relatively high, to encourage the exploration of distant neighbors. But over the iterations, this value decreases, so that the algorithm is progressively pushed towards exploitation by selecting ever closer neighbors. Although this mechanism aims at an adaptive balance, an overly rapid decrease reduces the diversity of solutions, and limits the ability of the algorithm to escape from locally optimal zones.



(a) Stochastic variability induced by random sampling of neighbors

(b) Transition from exploration to exploitation based on R/j and loss of diversitye

Figure 6. Limitations of the EFACO approach

As explained in Fig. 6, present clearly the limitations of the EFACO algorithm. Sub-figure (a) shows that the random selection of neighbors generates significant stochastic variability, making neighborhood exploration inefficient and leading to pheromone dispersion that slows convergence. This instability makes it difficult to find robust solutions and undermines the reproducibility of results. Sub-figure (b) shows the dynamics of the transition from exploration to exploitation: as iterations progress, the diversity of solutions decreases rapidly while exploitation dominates, increasing the risk of premature stagnation in local optimal. As a result, without adaptive regulation mechanisms, the balance between exploration and exploitation is difficult to balance, and limits the capacity of the algorithm to efficiently optimize the composition of QoS-aware services in large scale.

In order to tackle the weaknesses of the EFACO algorithm, notably the stagnation risk and the variability due to random decisions, we propose a new approach inspired by the hierarchical structure of the GWO. In this approach, the three leader ants of each iteration are assimilated to the dominant individuals of the swarm ϑ, μ and σ , and their

influence is weighted into the overall pheromone updating process (as illustrated in the equation 16). This hierarchy ensures that exploration focuses on the most promising regions of the search space, but preserves the diversity to avoid convergence towards local optimum. In addition, a local redistribution of pheromones to neighboring services ensures that high-quality information is distributed in the vicinity, reinforcing local exploration and contributing to the stability of the optimization process. This double dynamic, both global and local, allows the algorithm to achieve an efficient balance between intensification and diversification.

This work proposes a new global pheromone update based on the weighted average influence of the three best ants, identified as ϑ, μ and σ . Inspired by the leadership principle in the GWO, this strategy spreads the influence of the best solutions in a more balanced and robust way. The following equation formalizes this mechanism, which combines the evaporation effect with proportional reinforcement based on the relative quality of solutions by leaders.

$$\tau_{i,j}(t+1) = (1-\rho)\,\tau_{i,j}(t) + \rho \sum_{k\in\{\vartheta,\mu,\sigma\}} w_k \cdot \Delta\tau_{i,j} \tag{16}$$

where ρ is the evaporation rate, $\Delta\tau_{i,j}$ represents the amount of pheromone deposited by each leader ant, and $w_k \in [0,1]$ is the weight assigned to each ant based on its solution quality ($w_\delta > w_\mu > w_\sigma$). This formula takes into account several high-quality solutions to guide the search process, rather than relying solely on a single best ant. As a result, it improves convergence stability and mitigates premature stagnation to maintain diversity in the search dynamic.

After the completed tour, each leader ant $k \in \{\vartheta, \mu, \sigma\}$ disposes an amount of pheromone $\Delta\tau_{i,j}^{(k)}$ on an arc (i,j) during the global update.

$$\Delta\tau_{i,j}^{(k)} = \begin{cases} \dfrac{Q}{L_k}, & \text{if } (i,j) \in L^{(k)}, \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

where $L^{(k)}$ denotes the solution built (path founded) by leader ant k, L_k is the total cost of this solution (in terms of time, cost, reliability, etc.) and Q is a positive deposition constant fixed in advance, and $\Delta\tau_{i,j}^{(k)}$ corresponds to the amount of pheromones deposited by k on arc (i,j).

To avoid the problem of selecting neighboring services randomly or in a static way in order to inject pheromones, in this work we propose to inject services on a dynamic and harmonized manner. Thus, this injection is made proportionally according to the frequency of services belonging to the solution found by ant leader: the least frequent services receive more reinforcement in order to encourage exploration and increase their chances of being selected in future iterations, while the most frequent services inject a progressively reduced amount in order to avoid search stagnation. This proportional distribution allows us to better distribute the influence of services, to maintain a good balance between exploration and exploitation, and to improve the quality of the solutions found.

After the global update phase, each leader ant injects a number of pheromones into the nearest neighbors according to the frequency of services. The equation below describes how pheromones are distributed to neighboring nodes:

$$\tau(t+1,s)(t+1,l) = \tau(t+1,s)(t+1,l) + \left(\frac{\tau(t,s')(t+1,s)}{e^{\alpha\cdot d'(\eta(s,l))}}\right) \times \left(\frac{\phi(s)}{\sum_{q\in S(k)}\phi(q)}\right) \times w_k \tag{18}$$

Where $l \in [1, Nh]$ is a service neighboring s in the space of services available to the task T_{t+1}. Nh is the number of neighbors. $\tau(t,s')(t+1,s)$ indicates the pheromone tracks from the global pheromone update. $d'(\eta(s,l))$ represents the normalized distance between the services of task $i+1$ and its neighbor, Web service l, in the same task. $\phi(s)$ denotes the functional weight of service s, calculated from its frequency of appearance in previous solutions, so that less frequent services receive a higher weight. This weight is normalized by the sum $\sum_{q\in S(k)}\phi(q)$, where $S^{(k)}$ is the set of services presents in the leader's solution k, thus ensuring that the pheromone injection is proportional to the relative importance of each service in the overall solution.

In order to adjust the impact of each service according to how often it appears in the previous solutions (as illustrated in fig 7), we define a functional weight $\phi(s)$. This weight is designed to encourage exploration around infrequent services while maintaining minimal influence for dominant services. It is calculated by formula below :

$$\phi(s) = e^{-\chi_s/\chi_{max}} \tag{19}$$

Let $\chi_s \in \mathbb{N}$ denote the number of occurrences of service s in the solutions explored up to the current iteration. The value χ_{max} represents the maximum frequency observed across all candidate services. These quantities are used to normalize the frequency of a service to weight its importance in pheromone injection.

The dynamic number of neighbors $N_h^{(k)}$ that each leader can reinforce, based on the quality of its solution, is determined by formula below. This quantity, derived from a logarithmic normalization between the solution cost for the leader L_k and the maximum cost $L_{\max}$, allows us to adjust automatically the exploration intensity allocated to each solution. This total number of neighbors is then distributed between services $s \in S^{(k)}$, proportionally to their inverse frequency of occurrence, modeled by a weight $\phi(s)$. This distribution is defined by formula 21, which ensures that each service has a minimum number of reinforcements $N_{\min}$, while concentrating injection on services that are least used. All this ensures harmonized regulation between local exploitation of dominant services and stimulus targeted at under-valued alternatives.

$$Nh^{(k)} = |T_{i+1}| \cdot \frac{\log(1 + L_k)}{\log(1 + L_{\max})} \tag{20}$$

$$Nh_{t+1}^{(k)}(s) = \max\left(N_{\min}, \left\lfloor Nh^{(k)} \cdot \frac{\phi(s)}{\sum_{q \in S^{(k)}} \phi(q)} \right\rfloor\right) \tag{21}$$

In addition, the equation 21 determines the maximum number of neighbors that each selected service in a solution leader is allowed to reinforce, based on the number of neighbors it has historically contributed to previously explored trajectories. This is derived from the number of times the service has appeared in previous iterations, through a functional weight $\phi(s)$. The distribution is carried out proportionally, ensuring that services that are weakly exploited in the search memory have greater injection power. A minimum threshold $N_{\min}$ is introduced to maintain a guaranteed minimum influence even for strongly used services.

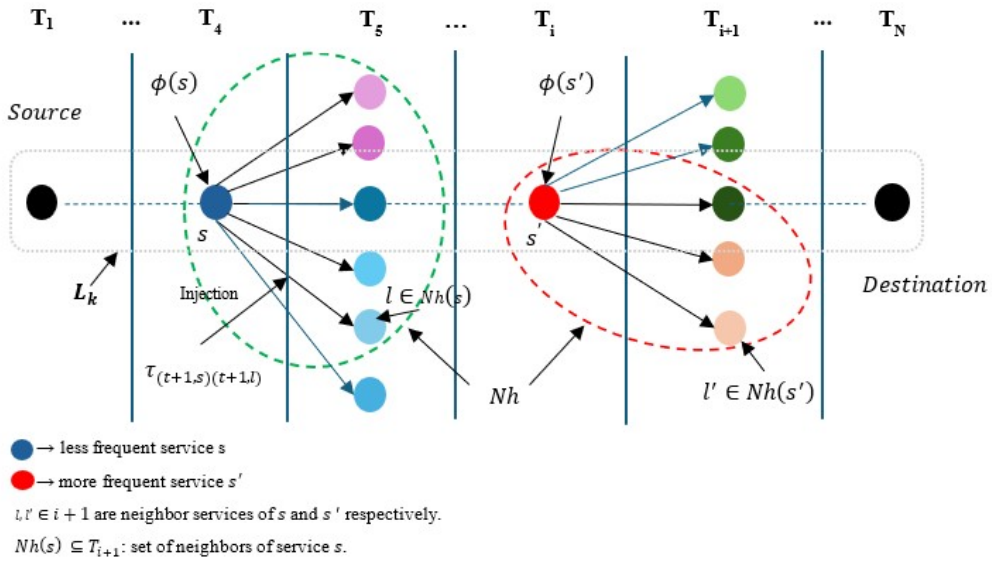


Figure 7. An example of pheromone injection process based on service frequency

In the fig.7 illustrates the injection of pheromones from two services s and s' belonging to the solution of a leader L_k. The number of reinforced neighbors depends on the functional weight $\phi(s)$, proportional to the dearth of the

service. Less frequent services (left) explore a larger neighborhood, while more frequent services (right) have limited influence.

This pheromone injection strategy, and the selection a maximum number of neighboring services to be reinforced, offers two main objectives:

— Firstly, it targets the neighboring services most similar in function to those found in the solution, while ensuring a harmonized distribution of pheromones. This means the less frequent services receive more reinforcement, encouraging exploration in future iterations, while the more frequent services maintain a modest influence, to avoid excessive domination.

— Secondly, this approach reduces the execution time, as it restricts injection to the most relevant neighbors rather than randomly selected or injected all available services, which improves both computational efficiency and the quality of injection decisions.

4.2. Dynamic parameters adjustment

Most variants of ACO algorithms such as EFACO, FACO, control parameters such as α, β, and ρ are defined a priori as fixed and constant values throughout the running of the algorithm. These parameters are often selected empirically or by cross-validation on a training dataset. These three hyperparameters play an important role to control the balance between exploitation of the acquired knowledge and exploration of new solutions. Nevertheless, when held constant throughout the runtime, these parameters can give rise to certain main limitations, but classifying them into three cases, are as follows:

— **First case:** if $\beta > \alpha$, this translates that our first iteration favors local heuristic exploration, which is preferable at the beginning of the process of the algorithm, but if it continues in the last iterations, it can lead to stagnation holding the algorithm in locally optimal or sub-optimal solutions, without allowing efficient convergence to the optimal solution.

— **Second case**: if $\alpha > \beta$, this means that the algorithm favors the exploitation of pheromone-marked solutions from the outset. This can help accelerate convergence, but if exploitation is excessive from the first iterations, the algorithm could prematurely get locked into a locally optimal solution, and neglect the exploration of other more promising tracks. This approach reduces the diversity of solutions and may prevent the optimal solution from being reached.

— **Third case:** when the evaporation rate ρ is badly adjusted (too low or too high), this disrupts the memory management mechanism. We take two scenarios where:

- ρ is too low, previous pheromone traces become active for a long time, leading to excessive memory accumulation. This can artificially reinforce sub-optimal paths explored early to reduce the adaptive capacity of the system.
- Conversely, when ρ is too high, the pheromones quickly evaporate, preventing the algorithm from capitalizing on the promising trajectories already identified. This makes the process too unstable and prevents robust convergence.

In addition, these parameters have a major impact on the transition probability, which means that whatever the quality of the population generated at each iteration, the transition probabilities are always calculated using the same exponential weights. On the other hand, this prevents the algorithm from reinitializing the search in the event of prolonged stagnation, leading to insufficient detection of solution diversity.

In order to address these limitations and improve convergence in a balanced way while maintaining solution diversity, this section proposes a dynamic self-adaptive adjustment mechanism for α, β, ρ, that relies on the quality of the solutions generated to answer the question of which phase (exploration or exploitation) should be included in the next iteration.

This mechanism aims to improve the ability of the algorithm to avoid local optimum, to encourage convergence towards globally promising areas, while maintaining an intelligent balance between diversification and intensification.

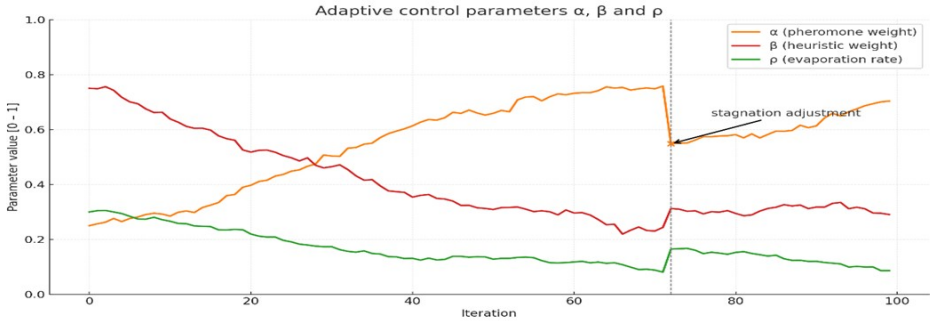


Figure 8. Adaptive Adjustment of ACO Parameters: An Illustrative Example (Stagnation Problem)

Fig. 8 shows how the adaptive control mechanism for the parameters α, β, and ρ efficiently surmounts the stagnation problem while maintaining a dynamic balance between the two phases mentioned in the previous paragraph. During the first iterations (iteration = 72), the progressive increase in α and decrease in β reflect a phase of intensification guided by the best solutions explored. However, at iteration, stagnation is detected, the controller automatically reacts by reducing α, increasing β and ρ, thus restarting the exploration process. This adaptive reset enables the algorithm to avoid a sub-optimal trap and converge back to better solutions, as evidenced by the gradual rise in α and the stabilization of the parameters, demonstrating the system's ability to dynamically adjust its behavior based on observed progress.

In the DPACO algorithm, the three parameters of the algorithm are determined dynamically, based on the analysis of the relative evolution of the quality of the solutions. To determine these parameters, at each iteration t, we evaluate the quality evolution of the best solution F^t_{best} compared to the previous best solution F^{t-1}_{best}. The relative rate of improvement is defined as:

$$\delta^t = \frac{F^{t-1}_{\text{best}} - F^t_{\text{best}}}{F^{t-1}_{\text{best}}} \tag{22}$$

This rate represents the variation in performance between two successive iterations. A $\delta^t > 0$ indicates improvement and $\delta^t \leq 0$ reflects stagnation or degradation.

At each iteration, the three main ACO parameters are updated as follows:

$$\begin{aligned} \alpha^{t+1} &= \alpha^t \cdot (1 + \delta^t) \\ \beta^{t+1} &= \beta^t \cdot (1 - \delta^t) \\ \rho^{t+1} &= \rho^t \cdot (1 - \delta^t) \end{aligned} \tag{23}$$

This mechanism allows us to dynamically adjust the balance between exploration and exploitation based on the solution quality and performance achieved during optimization. To avoid unexpected variations in parameters, we introduce a progressive adjustment mechanism. This approach combines previous parameter values with new updates, which allows for a controlled evolution over iterations. This progressive adjustment ensures that the algorithm maintains a stable dynamic, capable of efficiently adapting to changes while ensuring reliable convergence.

$$\theta^{t+1} = \gamma \cdot \theta^t + (1 - \gamma) \cdot \theta^{t+1} \tag{24}$$

where $\theta \in \{\alpha, \beta, \rho\}$ and $\gamma \in (0, 1)$ is the inertia factor that controls the proportion of previous values. This ensures a smooth and gradual transition, reinforcing the stability of the algorithm.

Although smooth adjustment stabilizes the evolution of the parameters, it is necessary to monitor the diversity of the population in order to adjust the balance between the diversification phase and the optimal selection phase.

Overly low diversity can lead to early convergence towards a local optimum, while excessive dispersion can lead to slower progress. For this, we use the standard deviation of costs, calculated by:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(F_i - \overline{F}\right)^2} \tag{25}$$

where N is the number of solutions, F_i their individual cost, and $\overline{F}$ the mean. This indicator evaluates the dispersion of solutions and provides guidance for adjusting the parameters.

With dynamic parameter adjustment, the algorithm can adjust dynamically to variations in the search space. The result is greater stability in the optimization process, avoiding stagnation and dependency on fixed settings, and more efficient progress towards high-quality, optimal solutions. This dynamic flexibility is essential to optimize highly complex search environments which are constantly evolving.

4.3. Complexity Analysis

The DPACO algorithm is based on the following steps, which are performed at each iteration. These steps are taken into account to analyze the time and space complexity of the proposed approach, as each one has a specific impact on the overall computational cost. Therefore, it is necessary to study them in order to evaluate the impact of the adaptive mechanisms introduced by DPACO compared to variants of the ACO algorithm, such as EFACO.

As indicated in Algorithm 1, the first step of DPACO is to construct solutions using a population of S ants. At each iteration, each ant makes n successive decisions to select a service for each of the workflow tasks, based on a probabilistic transition rule using pheromone and heuristic information. Since the calculation of this probability and the associated selection are performed in constant time, the cost of this step is proportional to the number of ants and the number of tasks, resulting in a complexity of $O(S \cdot n)$ per iteration.

During the construction phase, local pheromone updates are applied in order to immediately integrate the decisions made by the ants during task transitions. As these updates directly relate to the successive decisions taken by each ant, their cost is proportional to the number of transitions made. Consequently, this step has linear complexity and doesn't have an impact on the order of the solution-building phase.

After all solutions have been constructed, the algorithm selects the best solutions found in the current iteration. For each leader, a global update of pheromones is applied to reinforce the transitions associated with its solution. These operations are based on the search for solutions limited in size by the number of tasks n. Since the number of leaders is constant, the cost of this phase remains linear and can be estimated at $O(n)$ per iteration.

In the next phase, the algorithm injects pheromones into the neighbors of the leader solutions. For each leader and for each of the n tasks in its solution, a set of N_h neighboring services is selected from among the candidate services for the next task $t+1$. Pheromones are then injected from each services that make up the solution of the leader to its neighboring services associated with this task. Given that the injection is performed for n tasks and N_h neighbors per task, this phase has, in the worst case, a complexity of $O(n \cdot N_h)$ per iteration. In practice, the value of N_h is controlled, which limits the computational cost of this step.

Finally, the algorithm dynamically updates the control parameters α, β, and ρ in order to adapt the search behavior during iterations. This step is based on simple calculations that are independent of the number of tasks and the number of candidate services, resulting in a constant cost per iteration, denoted $O(1)$.

Considering all the steps described previously and taking into account Z iterations, the overall time complexity of the DPACO algorithm is dominated by the construction of solutions and the mechanism for injecting pheromones to neighbors. It can therefore be expressed as $O(Z \cdot n \cdot (S + N_h))$. Furthermore, the space complexity is mainly related to the storage of pheromones and solutions generated by the ant population, which leads to a memory cost of the order of $O(n \cdot m + S \cdot n)$.

With respect to EFACO, DPACO shares the same asymptotic complexity, as both algorithms are based on the same ACO-based solution construction mechanism, which remains the dominant component of computational cost. The main difference is in the injection of pheromones: while EFACO diffuses pheromones to all candidate services in the neighborhood, DPACO limits this injection to a controlled number of neighbors N_h, associated with the

services of the next task and applied only to leaders. This mechanism reduces redundant calculations, with the additional cost being bounded by $\mathcal{O}(n \cdot N_h)$, where N_h is much smaller than the total number of candidate services, without increasing the overall complexity.

Algorithm 1 DPACO: Dynamic Parameter Ant Colony Optimization

Inputs:
$T = \{t_1, \ldots, t_n\}$: set of tasks
S_i: candidate services for each task
S: number of ants, Z: maximum number of iterations
$\alpha, \beta, \rho, \tau_0$: algorithm parameters
Output:
S_{best}: best executable composition plan
Initialize pheromone trails $\tau \leftarrow \tau_0$
Initialize parameters α, β, ρ
Initialize $S_{best} \leftarrow \emptyset$
for $iter = 1$ to Z **do**
    for $k = 1$ to S **do**
        Randomly select a service for task t_1
        for $j = 2$ to n **do**
            Select next service using Eq. (5)
            Apply local pheromone update using Eq. (9)
        end for
    end for
    Select leader ants ϑ, μ and σ
    Update global pheromone trails using Eq. (16)–(17)
    for all $k \in \{\vartheta, \mu, \sigma\}$ **do**
        Compute the number of neighbors $N_h(k)$ using Eq. (20) service s in the leader solution
        for $l = 1$ to $N_h(k)$ **do**
            Inject pheromone into neighboring service l
            using Eq. (18), Eq. (19) and Eq. (21)
        end for
    end for
end for
Update α, β and ρ dynamically using Eq. (22)–(25)
Update S_{best} if a better solution is found

return S_{best}

5. Experimental study

This section covers two main parts. First, we will present in detail the hardware and software components used, the datasets selected, and the parameters configured for the algorithms selected for comparison. Secondly, an in-depth discussion will be devoted to analyzing the results obtained by the proposed solution, comparing them with those obtained by existing approaches, in order to highlight the contributions and limitations of our proposed method.

5.1. Experimental settings

5.1.1. Setup considerations

To evaluate the performance and efficiency of DPACO and the reference methods, we have implemented all the algorithms in Python. All experiments were carried out on a Dell G15 laptop equipped with an Intel® Core™ i5-11400H processor (6 cores, 2.7 GHz base frequency), 16 GB DDR4 RAM and an NVIDIA GeForce RTX 3060 graphics card, running Windows 11. The complete hardware and software setup used for the comparative experiments is detailed in Table 3.

Table 3. Hardware and software features

Hardware and Software	**Technical characteristics**
Implementation Language	Python
Processor (CPU)	Intel® Core™ i5-11400H (6 cores, 2.7 GHz)
Memory (RAM)	16 GB DDR4
Graphics Card (GPU)	NVIDIA GeForce RTX 3060
Operating System	Windows 11

With regard to the control parameters, in order to standardize the experimental conditions for all algorithms, we have used a range of common values, mainly based on the literature, notably those proposed in [32], and adapted to ensure experimental homogeneity. Although identical parameters were adopted for all algorithms, some 40 initial experiments were carried out to validate and refine these values. This process ensured that the chosen parameters offered a good balance between solution quality and execution time, to ensure a reliable and appropriate comparison. Table 4 presents the parameter values adopted for all the algorithms considered.

Table 4. Experimental setup: Control parameters used for all algorithms

Parameter	**Value**
Number of iterations (t)	100
Number of ants (m)	30
Pheromone initialization (τ_0)	0.1
Pheromone factor (α)	1
Heuristic factor (β)	2
Evaporation rate (ρ)	0.9

5.1.2. Dataset

In order to compare our approach with the other algorithms mentioned in the previous section, we have used our dataset, which is an extended version of the one used in the EFACO article. [32]. This extension not only includes the two original experimental scenarios (varying the number of tasks or the number of services), but also introduces a third mixed scenario, in which both dimensions vary simultaneously. Furthermore, we considered the same QoS attributes as in the EFACO experiments, namely response time, availability, reliability, and cost, to ensure a fair and coherent setting for comparison.

Specifically, our dataset is organized into three experimental scenarios, as illustrated in Table 5:

- **Scenario 1:** The number of tasks is fixed at 10, while the number of candidate services per task varies from 100 to 1000, in increments of 100. This scenario allows us to analyze the impact of increasing service choice per task on algorithm performance.
- **Scenario 2:** The number of candidate services is fixed at 100, while the number of tasks varies from 10 to 100, in increments of 10. This evaluates scalability with respect to increasing workflow size.
- **Scenario 3:** Both parameters vary simultaneously. The number of tasks is set to 10, 20, and 50, and for each of these, the number of services varies among 100, 500, and 1000, resulting in nine dataset combinations. This mixed configuration simulates more complex and realistic service composition environments.

With respect to the dataset used in the EFACO, which is limited to the exploration of two separate experimental dimensions (variation in the number of tasks or the number of services), our dataset offers a richer and more flexible evaluation setting. Scenario 1 follows the basic EFACO configuration, varying the number of candidate services per task, while maintaining a fixed number of tasks. This allows us to evaluate the performance of the algorithms in a larger search space. Scenario 2 explores how the opposite effect can be achieved, by progressively increasing the number of tasks in the workflow, with a constant number of services, in order to observe how the algorithm behaves when faced with longer compositions. Finally, Scenario 3 surpasses the limits of the EFACO protocol by simultaneously varying the number of tasks and the number of services, to simulate more complex and dynamic environments that are similar to real cases. This structure enables a more in-depth and robust analysis of algorithm performance under a wide range of conditions.

Table 5. Overview of the Experimental Dataset Scenarios

Scenario	**Number of Tasks**	**Number of Services per Task**
Scenario 1	Fixed at 10	100 to 1000
Scenario 2	10 to 100	Fixed at 100
Scenario 3	10, 20, 50	100, 500, 1000

5.2. Experimental Results and Discussion

5.2.1. Results and Comparison

To evaluate the effectiveness of our approach and compare to existing algorithms, we carried out a series of experiments based on the dataset discussed previously. This section presents and analyzes the results obtained

in terms of solution quality, execution time and convergence, comparing our algorithm with the main concurrent approaches in different experimental scenarios.

As a first series of experiments, we evaluated the performance of our approach in comparison with other existing algorithms, using a neighbor selection strategy and static parameters. This aims to show the impact of intelligent neighbor selection in a classical experimental setting. In a second series of experiments, we analyze the impact of dynamic adjustment of the control parameters on the quality of the solutions produced, in order to demonstrate the effectiveness of our adaptive algorithm to more complex and variable environments. The results of these two series of experiments are presented and analyzed in the rest of this section.

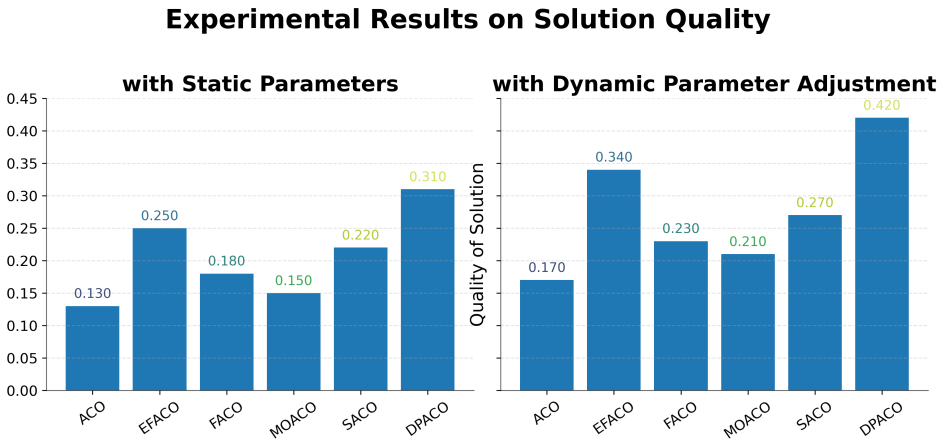


Figure 9. Experimental Results of Different ACO Variants on Solution Quality

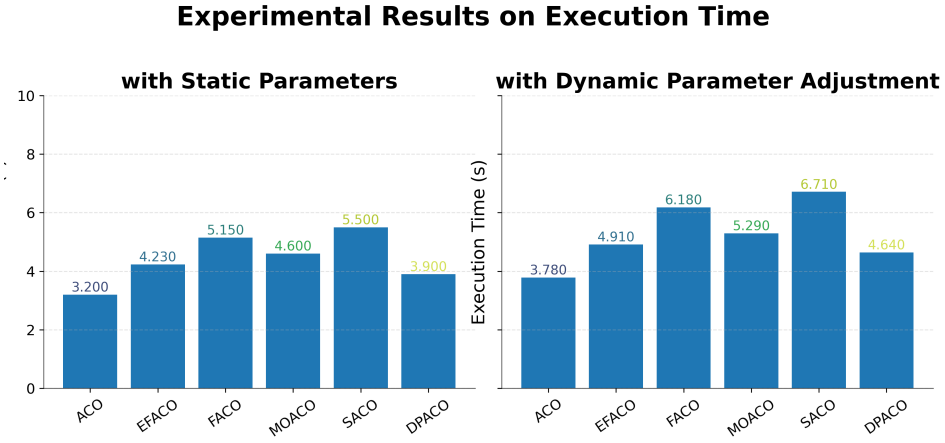


Figure 10. Experimental Results on Execution Time of ACO Variants

The experimental results illustrated in the two figures 9 and 10 demonstrate how the DPACO approach outperforms existing algorithms. As regards solution quality, with dynamic adjustment and harmonized injection, DPACO achieved a score of 0.420, which surpassed EFACO (0.340), FACO (0.230), MOACO (0.210), SACO (0.270), and significantly higher than ACO (0.170). With regard to execution time, DPACO is up from 3,900 s to 4,640 s in the dynamic version. This is due to the time required for additional calculations as parameters are continuously adjusted. Nevertheless, it is still lower than the execution times observed in other dynamic variants, such as FACO (6,180 s),

SACO (6,710 s) and MOACO (5,890 s). These results show that DPACO offers a good balance between the quality of the solutions obtained and the required computation time, which makes it suitable for environments where both accuracy and efficiency are important.

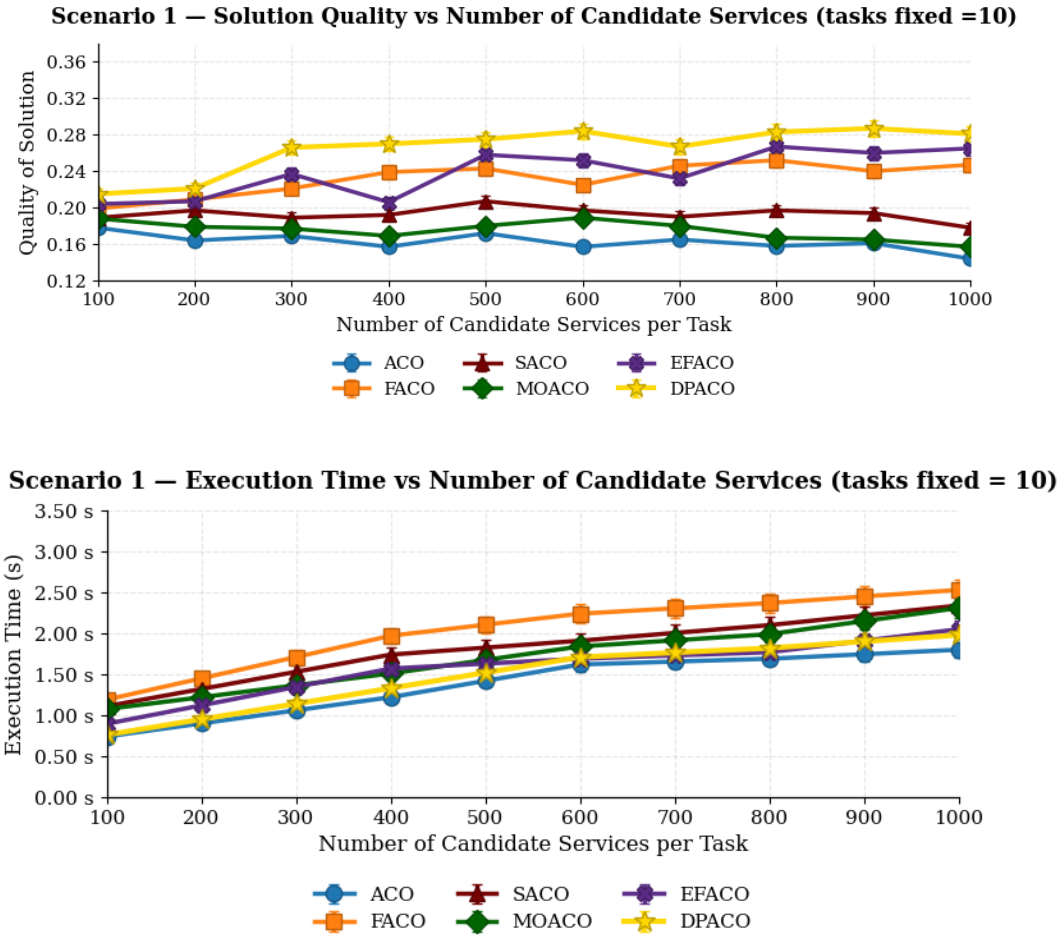


Figure 11. Benchmarking of the six ACO variants: solution quality and execution time according to the candidate services (Scenario 1, 10 tasks fixed)

In Scenario 1 (number of tasks set at 10), Figure 11 indicates clearly that the proposed DPACO algorithm achieves the best results in terms of solution quality when the number of candidate services per task increases gradually from 100 to 1000. The average quality obtained by DPACO is between 0.24 and 0.29, significantly better than EFACO and FACO (between 0.18 and 0.22), as well as classic variants such as MOACO, ACO, and SACO, which remain below 0.19. This performance is due to the combination of a dynamic and smart injection of pheromones to neighbors and an automatic algorithm parameter adjustment, which lets DPACO adapt in real time to changes in the problem. This strategy strengthens the balance between exploration and exploitation, and reduces stagnation. These results confirm the robustness, scalability, and adaptive capacity of DPACO in the face of an increasing number of available services.

In Scenario 2 (number of candidate services set at 100), Figure 12 reinforces these observations by evaluating the performance of the algorithms as the number of tasks gradually increases from 10 to 100. The composition workflow then becomes more and more complex, with an explosive increase in the number of possible configurations. Despite

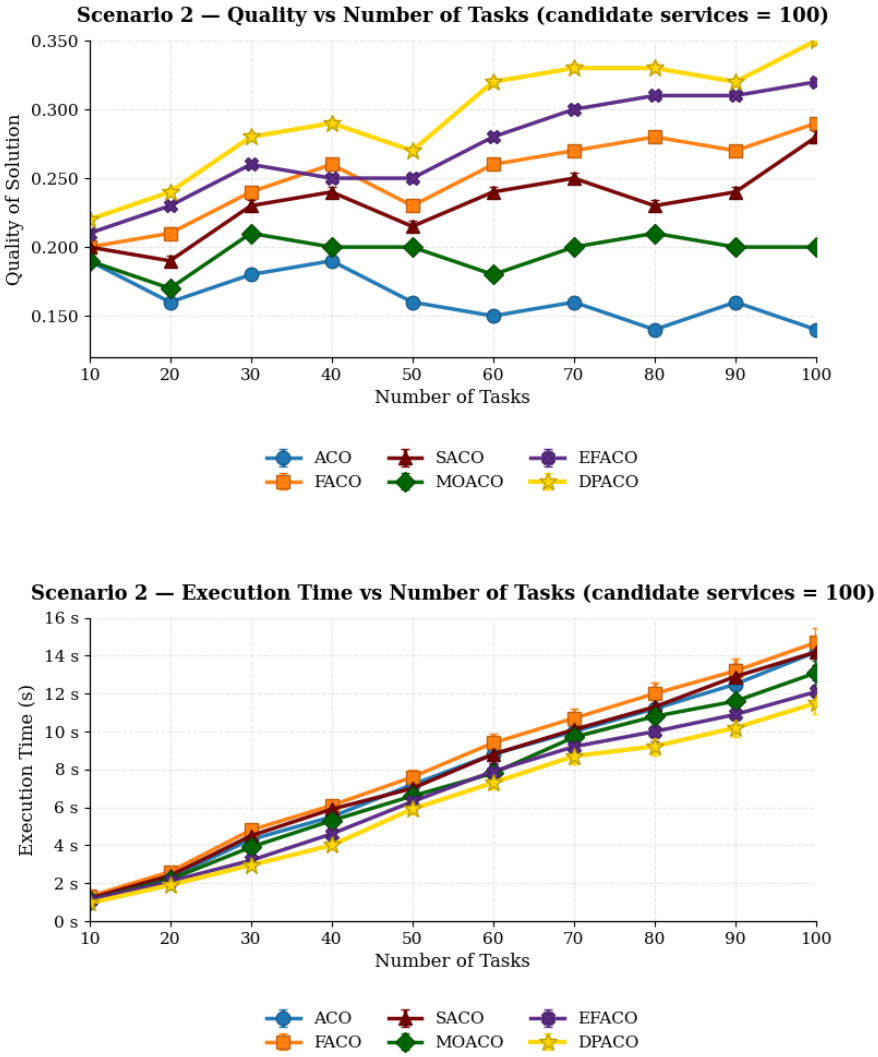


Figure 12. Benchmarking of the six ACO variants: solution quality and execution time according to the number of tasks (Scenario 2, 100 candidate services)

this, DPACO maintains a high solution quality, as high as 0.33, compared to 0.26 to 0.30 for EFACO, 0.21 to 0.28 for FACO, and less than 0.21 for MOACO and ACO. With dynamic parameter tuning and smart pheromone deposition management, DPACO is able to maintain efficient exploration while reducing unnecessary computational overhead. At the same time, execution time remains under control, between 2.1 s and 13.5 s, lower than FACO (up to 15.4 s) and close to EFACO, confirming that DPACO adapts effectively to increasing complexity, with an optimal balance between performance and accuracy.

In Figure 13, the results show that DPACO confirms its performance in terms of solution quality. In terms of quality, DPACO achieves high scores, ranging from 0.348 (T10–S100) to 0.397 (T10–S1000), and maintains robust quality even in the most complex cases such as T50–S1000 (0.393). By comparison, in this same configuration, ACO

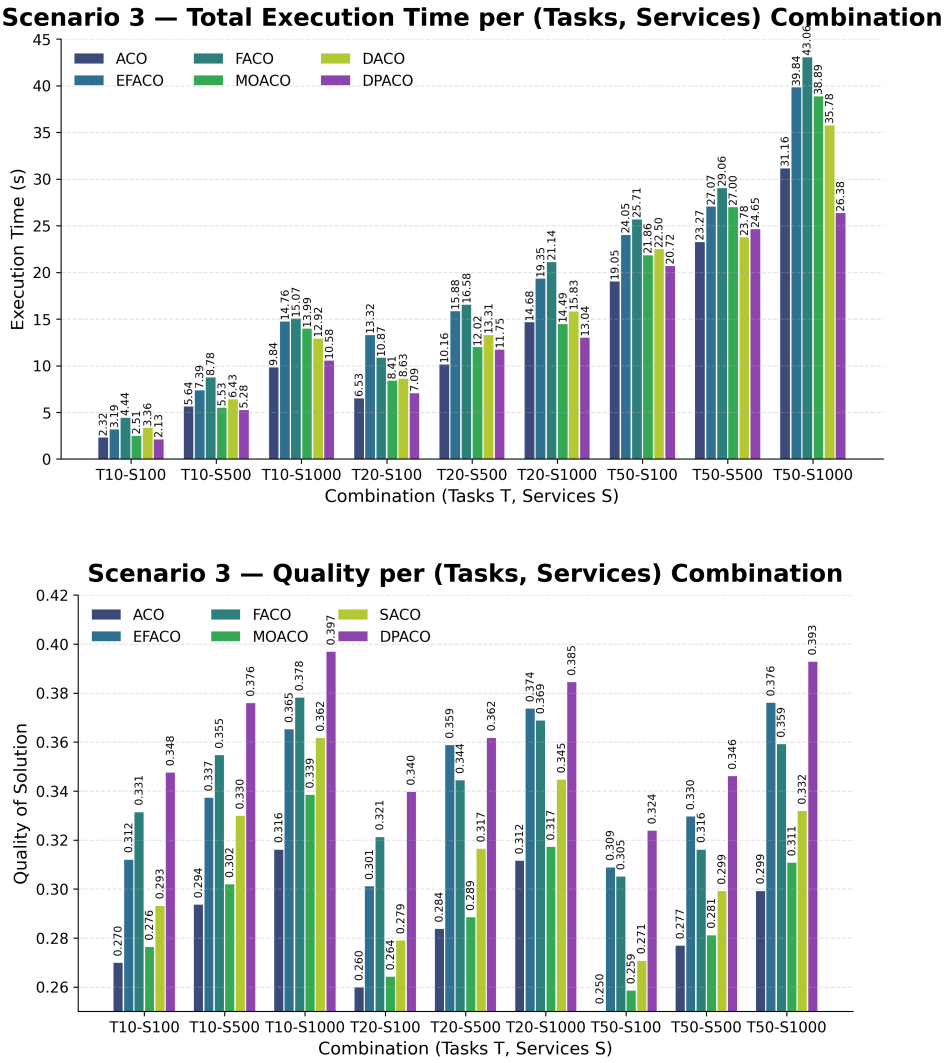


Figure 13. . Benchmarking of the six ACO variants: total execution time and solution quality according to task-service combinations (Scenario 3).

and FACO achieve 0.282 and 0.376 respectively, while SACO remains at 0.352. These results show that DPACO is capable of providing high-quality solutions, even when the number of tasks and services increases significantly.

As regards execution time, DPACO also demonstrates stability. It maintains a processing time of less than 26 seconds in all combinations tested, with a minimum of 3.29 s (T10–S100) and a maximum of 25.23 s (T50–S1000). By comparison, in the most demanding case, FACO reaches 43.06 s, ACO 39.34 s, and SACO 36.45 s. These differences demonstrate the ability of DPACO to effectively manage increasing complexity without resulting in an explosion in time costs.

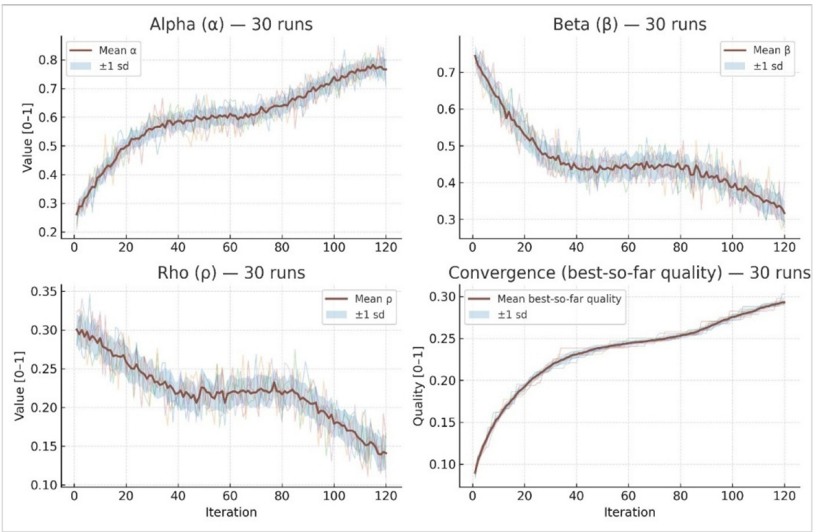


Figure 14. Performance evaluation of the adaptive parameter adjustment on 30 runs

To validate the effectiveness of our dynamic parameter adjustment approach, Figure 14 shows the adaptive evolution of the parameters α (alpha), β (beta), and ρ (rho), as well as the convergence of the solution quality over iterations, based on 30 independent runs of our DPACO algorithm. Reportedly, the parameter α, representing the importance of the pheromone trail, gradually increases to reach an average of 0.78 around the 120th iteration, thus reflecting a growing orientation towards the exploitation of promising paths. At the same time, the parameter β, which weights the local heuristic, decreases steadily from 0.72 to around 0.36, reflecting a gradual reduction in the influence of heuristics in favor of pheromone-based learning. The evaporation rate ρ shows a decreasing trend, ranging from 0.33 to 0.13, and indicates a gradual stabilization of pheromone deposition. Finally, the convergence curve shows a steady improvement in the best quality achieved, rising from 0.12 to 0.29 on average, with low dispersion (±0.01), confirming the stability and effectiveness of the adaptive mechanism integrated into DPACO. This dynamic regulation allows the algorithm to intelligently adjust to the progress of the search, ensuring an effective compromise between initial exploration and convergence towards optimized solutions.

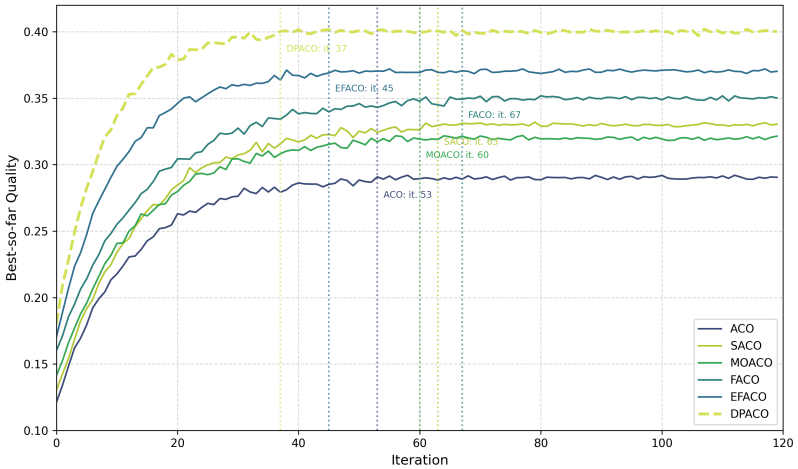


Figure 15. Convergence Curve on T50-S500 of all variants of ACO

As shown in the figure 15, the convergence curves of the different ACO algorithms are illustrated in terms of "best-so-far" quality over iterations. The proposed version, DPACO, differs significantly in terms of convergence speed. In practical terms, it achieves a stable solution after just 37 iterations, which demonstrates its effectiveness in quickly exploring the search space and exploiting promising solutions. This is followed by EFACO, which converges at the 45th iteration, while ACO, SACO, MOACO, and FACO reach their plateaus at the 53rd, 60th, 60th, and 67th iterations, respectively.

This rapid rate of stabilization reflects an ability to avoid redundant explorations while maintaining a high-quality solution. DPACO is therefore able to stand out as an efficient and robust approach, surpassing other variants in both precision and convergence speed.

5.2.2. Discussion and Limitations

The results obtained through various experiments highlight the superiority of the DPACO approach over traditional variants of the ACO algorithm. In particular, the performance observed in terms of solution quality, convergence speed, and stability demonstrates the value of dynamic parameter regulation. This strategy effectively strengthens the balance between exploration and exploitation over iterations, while adapting to the constantly evolving requirements of the search. In this section, we analyze these results in depth to better understand the factors contributing to this improvement and to identify the practical issues.

In order to evaluate the effectiveness of the proposed pheromone injection mechanism, we carried out a comparison between DPACO, EFACO, and FACO. As illustrated in Fig. 11, particularly with 400 candidate services, EFACO selects neighbors randomly, and leads to inefficient pheromone dispersion. FACO improves this process through static verification, but at the cost of increased rigidity. In contrast, DPACO adopts a dynamic and targeted strategy, better guiding exploration. This approach ensures faster and more stable convergence, with gains of approximately +11.6% over EFACO and +7.3% over FACO. This is confirmed in scenario 2 (Fig. 12), particularly with 40 tasks, where DPACO continues to dominate in terms of quality, while EFACO remains unstable and FACO more costly.

As the results in Fig. 13 demonstrate, DPACO achieves a stable and superior solution quality despite the simultaneous increase in the number of tasks and candidate services. This scalability can be explained by dynamic parameter adjustment and a targeted injection strategy, contrasting with the degradation observed in EFACO, FACO, and ACO. Furthermore, the low variance between similar configurations confirms the stability of the model in large-scale environments.

In addition, the results presented in Figures 14 and 15 confirm the positive impact of dynamic parameter control coupled with targeted pheromone injection in DPACO. This combination not only allows promising solutions to be reached more quickly than other variants, but also maintains better stability across all executions. By focusing intensification on relevant regions of the search space while adapting parameters as progress is made, DPACO effectively avoids stagnation in local optimal. This ability to balance exploration and exploitation enhances algorithmic robustness, particularly in complex and dynamic environments.

Limitations : Although dynamic parameter adjustment in DPACO reduces sensitivity to static settings in large-scale contexts, its effectiveness is still related to the evolution of solutions, which can affect the rate of adaptation of the mechanism when progress is slow. Furthermore, the interaction between adaptive parameters reflects the dynamic nature of the optimization process, and the trade-off between stability and adaptability remains naturally related to the problem context.

6. Conclusion

We have proposed a new approach called DPACO, in order to address the main constraints identified in the EFACO algorithm. Despite improvements over classical ACO variants, EFACO is still constrained by high computational costs, random neighbor selection, and insufficient scalability in dynamic environments. DPACO is based on the multi-objective principle of EFACO while incorporating several adaptive mechanisms. More specifically, our approach

introduces a dynamic, target-oriented pheromone deposit, adaptive local control to avoid stagnation and improve the balance between exploration and exploitation, reduced search time, and increased robustness in large-scale distributed environments.

In addition, experiments carried out on various datasets and 30 independent runs, our approach significantly outperforms ACO and its major variants (SACO, MOACO, FACO, EFACO), with gains of up to +15.2% in solution quality, while reducing convergence time. These results confirm the effectiveness of the proposed adaptive mechanisms and demonstrate that DPACO constitutes a robust advance in solving the problem of QoS-aware WSC. These results reinforce how important it is to include adaptive parameter control and dynamic pheromone injection in service composition that takes QoS into account. Future work could focus on the extension in hybrid optimization frameworks or its translation to other large-scale service-oriented environments, such as cloud or IoT ecosystems.

REFERENCES

1. Niknejad, N., Ismail, W., Ghani, I., Nazari, B., Bahari, M., & Hussin, A. R. B. C. (2020). Understanding Service-Oriented Architecture (SOA): A systematic literature review and directions for further investigation. Information Systems, 91, 101491. doi: https://doi.org/10.1016/j.is.2020.101491
2. Laskey, K. B., & Laskey, K. (2009). Service oriented architecture. Wiley Interdisciplinary Reviews: Computational Statistics, 1(1), 101-105. doi: https://doi.org/10.1002/wics.8
3. Papazoglou, M. P. (2008). Web Services: Principles and Technology. Pearson.
4. Velepucha, V., & Flores, P. (2023). A survey on microservices architecture: Principles, patterns and migration challenges. IEEE access, 11, 88339-88358. doi: 10.1109/ACCESS.2023.3305687
5. Górski, T., & WOźniak, A. P. (2021). Optimization of business process execution in services architecture: A systematic literature review. IEEE Access, 9, 111833-111852. doi: 10.1109/ACCESS.2021.3102668
6. Lemos, A. L., Daniel, F., & Benatallah, B. (2015). Web service composition: a survey of techniques and tools. ACM Computing Surveys (CSUR), 48(3), 1-41. doi: https://doi.org/10.1145/2831270
7. Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S., & Xu, X. (2014). Web services composition: A decade's overview. Information Sciences, 280, 218-238. doi: https://doi.org/10.1016/j.ins.2014.04.054
8. Gabrel, V., Manouvrier, M., & Murat, C. (2015). Web services composition: complexity and models. Discrete Applied Mathematics, 196, 100-114. doi: https://doi.org/10.1016/j.dam.2014.10.020
9. Garriga, M., Flores, A., Cechich, A., Zunino, A. (2015). Web services composition mechanisms: a review. IETE Technical Review, 32(5), 376-383. doi: https://doi.org/10.1080/02564602.2015.1019942
10. A. Zannou, A. Boulaalam and E. H. Nfaoui, "Predicting the Traffic Congestion and Optimal Route in a Smart City Exploiting IoT Devices," 2021 International Conference on Digital Age & Technological Advances for Sustainable Development (ICDATA), Marrakech, Morocco, 2021, pp. 53-57, doi: 10.1109/ICDATA52997.2021.00019.
11. Naoufal EL ALLALI, Mourad FARISS, Hakima ASAIDI and Mohamed BELLOUKI. "A Hybrid Similarity Measure for Dynamic Service Discovery and Composition based on Mobile Agents". International Journal of Advanced Computer Science and Applications (IJACSA) 12.12 (2021). doi: http://dx.doi.org/10.14569/IJACSA.2021.0121213
12. Zannou, A., Boulaalam, A., Nfaoui, E. H. (2022). Data Flow Optimization in the Internet of Things. Statistics, Optimization & Information Computing, 10(1), 93-106. https://doi.org/10.19139/soic-2310-5070-1166
13. Strunk, A. (2010, December). QoS-aware service composition: A survey. In 2010 Eighth IEEE European Conference on Web Services (pp. 67-74). IEEE. doi: 10.1109/ECOWS.2010.16.
14. Chattopadhyay, S., Banerjee, A. (2020). QoS-aware automatic web service composition with multiple objectives. ACM Transactions on the Web (TWEB), 14(3), 1-38. doi: https://doi.org/10.1145/3389147
15. Rodriguez-Mier, P., Mucientes, M., Lama, M. (2015). Hybrid optimization algorithm for large-scale QoS-aware service composition. IEEE transactions on services computing, 10(4), 547-559. doi: 10.1109/TSC.2015.2480396
16. Shehu, U., Epiphaniou, G., Safdar, G. A. (2014). A survey of QoS-aware web service composition techniques. International Journal of Computer Applications, 89(12), 10-17.
17. Jatoth, C., Gangadharan, G. R., Buyya, R. (2015). Computational intelligence based QoS-aware web service composition: a systematic literature review. IEEE Transactions on Services Computing, 10(3), 475-492. doi: 10.1109/TSC.2015.2473840
18. Jaeger, M.C., Mühl, G., Golze, S. (2005). QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms. In: Meersman, R., Tari, Z. (eds) On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE. OTM 2005. Lecture Notes in Computer Science, vol 3760. Springer, Berlin, Heidelberg. doi: https://doi.org/10.1007/11575771_41
19. Zeng, L., Benatallah, B., Ngu, A. H., Dumas, M., Kalagnanam, J., & Chang, H. (2004). QoS-aware middleware for web services composition. IEEE Transactions on software engineering, 30(5), 311-327. doi: 10.1109/TSE.2004.11.
20. El ALLALI, N., FARISS, M., ASAIDI, H., BELLOUKI, M. (2020, October). Semantic web services composition model using ant colony optimization. In 2020 Fourth International Conference on Intelligent Computing in Data Sciences (ICDS) (pp. 1-5). IEEE.doi: 10.1109/ICDS50568.2020.9268756.
21. Kouicem, A., Khanouche, M. E., Tari, A. (2022). Novel bat algorithm for QoS-aware services composition in large scale internet of things. Cluster Computing, 25(5), 3683-3697. doi: https://doi.org/10.1007/s10586-022-03602-6
22. El Allali, N., Fariss, M., Asaidi, H. et al. A web service composition framework in a heterogeneous environment. J Ambient Intell Human Comput 14, 12133–12157 (2023). doi: https://doi.org/10.1007/s12652-022-03761-9

23. Dahan, F. An Improved Whale Optimization Algorithm for Web Service Composition. Axioms 2022, 11, 725. doi: https://doi.org/10.3390/axioms11120725
24. Bhushan, S. B. & Reddy, P. C. (2018). A Hybrid Meta-Heuristic Approach for QoS-Aware Cloud Service Composition. International Journal of Web Services Research (IJWSR), 15(2), 1-20. doi: https://doi.org/10.4018/IJWSR.2018040101
25. Chen, J., & Zhou, J. (2020, December). An improved ant colony optimization for QoS-aware web service composition. In 2020 Eighth International Conference on Advanced Cloud and Big Data (CBD) (pp. 20-24). IEEE. doi: 10.1109/CBD51900.2020.00013.
26. Sbayti, O., & Housni, K. (2024). A new routing method based on ant colony optimization in vehicular ad-hoc network. Statistics, Optimization & Information Computing, 12(1), 167-181. doi: https://doi.org/10.19139/soic-2310-5070-1766
27. Du, Z., & Miao, H. (2022). An Optimization Method Based on Be-ACO Algorithm in Service Composition Context. Computational Intelligence and Neuroscience, 2022(1), 5231262. doi: https://doi.org/10.1155/2022/5231262
28. Alayed, H., Dahan, F., Alfakih, T., Mathkour, H., & Arafah, M. (2019). Enhancement of ant colony optimization for QoS-aware web service selection. IEEE Access, 7, 97041-97051. doi: 10.1109/ACCESS.2019.2927769.
29. Bei, L., Wenlin, L., Xin, S., & Xibin, X. (2024). An improved ACO based service composition algorithm in multi-cloud networks. Journal of Cloud Computing, 13(1), 17. doi: https://doi.org/10.1186/s13677-024-00588-x
30. Rabah, B., Mounine, H. S., & Ouassila, H. (2022). Qos-aware iot services composition: A survey. In Distributed Sensing and Intelligent Systems: Proceedings of ICDSIS 2020 (pp. 477-488). Cham: Springer International Publishing.
31. Kashyap, Neeti, Kumari, A. Charan and Chhikara, Rita. "Service Composition in IoT using Genetic algorithm and Particle swarm optimization" Open Computer Science, vol. 10, no. 1, 2020, pp. 56-64. doi: https://doi.org/10.1515/comp-2020-0011
32. F. Dahan, K. E. Hindi, A. Ghoneim and H. Alsalman, "An Enhanced Ant Colony Optimization Based Algorithm to Solve QoS-Aware Web Service Composition," in IEEE Access, vol. 9, pp. 34098-34111, 2021, doi: 10.1109/ACCESS.2021.3061738.
33. Dahan, F., El Hindi, K., & Ghoneim, A. (2017). An Adapted Ant-Inspired Algorithm for Enhancing Web Service Composition. International Journal on Semantic Web and Information Systems (IJSWIS), 13(4), 181-197. doi: https://doi.org/10.4018/IJSWIS.2017100109
34. F. Qiqing, P. Xiaoming, L. Qinghua and H. Yahui, "A Global QoS Optimizing Web Services Selection Algorithm Based on MOACO for Dynamic Web Service Composition," 2009 International Forum on Information Technology and Applications, Chengdu, China, 2009, pp. 37-42, doi: 10.1109/IFITA.2009.91.
35. Li, W., Yan-xiang, H. (2010). A Web Service Composition Algorithm Based on Global QoS Optimizing with MOCACO. In: Hsu, CH., Yang, L.T., Park, J.H., Yeo, SS. (eds) Algorithms and Architectures for Parallel Processing. ICA3PP 2010. Lecture Notes in Computer Science, vol 6082. Springer, Berlin, Heidelberg. doi: https://doi.org/10.1007/978-3-642-13136-3_22
36. Y. -m. Xia, J. -l. Chen and X. -w. Meng, "On the Dynamic Ant Colony Algorithm Optimization Based on Multi-pheromones," Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008), Portland, OR, USA, 2008, pp. 630-635, doi: 10.1109/ICIS.2008.112.
37. Wang, D., Huang, H., Xie, C. (2014). A Novel Adaptive Web Service Selection Algorithm Based on Ant Colony Optimization for Dynamic Web Service Composition. In: Sun, Xh., et al. Algorithms and Architectures for Parallel Processing. ICA3PP 2014. Lecture Notes in Computer Science, vol 8630. Springer, Cham. doi: https://doi.org/10.1007/978-3-319-11197-1_30
38. Chifu, V. R., Salomie, I., Pop, C. B., Niculici, A. N., & Suia, D. S. (2014). Exploring the selection of the optimal web service composition through ant colony optimization. Computing and Informatics, 33(5), 1047-1064.
39. Wu, Q., & Zhu, Q. (2013). Transactional and QoS-aware dynamic service composition based on ant colony optimization. Future Generation Computer Systems, 29(5), 1112-1119. doi: https://doi.org/10.1016/j.future.2012.12.010
40. Yu, Q., Chen, L., & Li, B. (2015). Ant colony optimization applied to web service compositions in cloud computing. Computers & Electrical Engineering, 41, 18-27. doi: https://doi.org/10.1016/j.compeleceng.2014.12.004
41. Dahan, F.; El Hindi, K.; Mathkour, H.; AlSalman, H. Dynamic Flying Ant Colony Optimization (DFACO) for Solving the Traveling Salesman Problem. Sensors 2019, 19, 1837. doi: https://doi.org/10.3390/s19081837
42. Zhao, C. Y., Wang, J. L., Qin, J., & Zhang, W. Q. (2014). A hybrid algorithm combining ant colony algorithm and genetic algorithm for dynamic web service composition. Open Cybern Syst J, 8, 146-154.doi: 10.2174/1874110X01408010146
43. Dahan, F. (2021). An effective multi-agent ant colony optimization algorithm for QoS-aware cloud service composition. IEEE Access, 9, 17196-17207. doi :10.1109/ACCESS.2021.3052907
44. S. Gohain and A. Paul, "Web service composition using PSO — ACO," 2016 International Conference on Recent Trends in Information Technology (ICRTIT), Chennai, India, 2016, pp. 1-5, doi :10.1109/ICRTIT.2016.7569553.
45. Du, Z., & Miao, H. (2022). An Optimization Method Based on Be-ACO Algorithm in Service Composition Context. Computational Intelligence and Neuroscience, 2022(1), 5231262. doi: https://doi.org/10.1155/2022/5231262
46. Yang, Y., Yang, B., Wang, S., Liu, F., Wang, Y., & Shu, X. (2019). A dynamic ant-colony genetic algorithm for cloud service composition optimization. The International Journal of Advanced Manufacturing Technology, 102(1), 355-368. doi: https://doi.org/10.1007/s00170-018-03215-7
47. Bei, L., Wenlin, L., Xin, S., & Xibin, X. (2024). An improved ACO based service composition algorithm in multi-cloud networks. Journal of Cloud Computing, 13(1), 17. doi:https://doi.org/10.1186/s13677-024-00588-x
48. Qi, L., Yao, W., & Chang, J. (2018, April). A large scale transactional service selection approach based on skyline and ant colony optimization algorithm. In NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium (pp. 1-7). IEEE. doi:10.1109/NOMS.2018.8406250
49. Vakili, A., Al-Khafaji, H. M. R., Darbandi, M., Heidari, A., Jafari Navimipour, N., & Unal, M. (2024). A new service composition method in the cloud-based internet of things environment using a grey wolf optimization algorithm and MapReduce framework. Concurrency and Computation: Practice and Experience, 36(16), e8091. doi:https://doi.org/10.1002/cpe.8091