Neural Computing and Applications (2024) 36:2901–2920
https://doi.org/10.1007/s00521-023-09195-4

ORIGINAL ARTICLE

Ant colony optimization for Chinese postman problem

Giacinto Angelo Sgarro[1] · Luca Grilli[1]

Received: 6 July 2023 / Accepted: 20 October 2023 / Published online: 25 November 2023

Abstract

This paper aims to solve the Chinese Postman Problem (CPP) using an Ant Colony Optimization (ACO) algorithm. In graph theory, the CPP looks for the shortest closed path that visits every edge of a connected undirected graph. This problem has many applications, including route optimization, interactive system analysis, and flow design. Although numerous algorithms aimed at solving CPP are present in the literature, very few meta-heuristic algorithms are proposed, and no ACO applications have been proposed to solve them. This paper tries to fill this gap by presenting an ACO algorithm that solves CPP (ACO-CPP). To prove its consistency and effectiveness, ACO-CPP is compared with a Genetic Algorithm (GA) and a recursive algorithm throughout three experiments: (1) recursive-ACO-GA comparisons over randomly generated graphs for the attainment of the global optimum; (2) ACO-GA statistical comparisons over specifically generated graphs; (3) recursive-ACO-GA comparisons by changing ACO hyperparameters over randomly generated graphs for the attainment of the global optimum. The experiments prove that the ACO-CPP algorithm is efficient and exhibits a consistency similar to GA when the number of possible solutions to explore is relatively low. However, when that number greatly exceeds those explored, ACO outperforms GA. This suggests that ACO is more suitable for solving problems with a CPP structure.

Keywords Ant colony optimization · Meta-heuristic · Chinese postman problem · Eulerian path · Vehicle routing problem

1 Introduction

The Chinese Postman Problem (CPP), for the first time proposed by the mathematician Kwan Mei-Ko in 1962 [1], is a problem that consists of finding the shortest route to deliver mail, traveling along every road and coming back to the post office [2].

From its first formulation onward, the CPP has been the subject of many real-life routing contexts, especially in urban applications such as garbage collection service, street sweeping, snow removal, sprinkling of salt on roads, network maintenance, security patrolling, and postal delivery service (mail carrier, newspaper boy) [3, 4].

To provide practical illustrations, real estate agents must traverse on foot through the streets of their area of interest within the city. They must navigate house by house to identify properties whose owners are interested in renting or selling as efficiently as possible. They aspire to accomplish this task in the shortest amount of time. In certain cities or specific sections, the need arises for pedestrian delivery to all residents, akin to the classical "newspaper boy problem". This involves manually sweeping streets and, in some cases, collecting waste and depositing it into carts. Such scenarios are particularly prevalent in historic city centers with narrow, winding streets. Not to mention street tourism, exemplified by a photographer who wishes to explore every thoroughfare in a quaint ancient village before deciding on subjects for photography. The underlying challenge across all these instances is solving the Chinese Postman Problem (CPP).

The CPP belongs to the Vehicle Routing Problem (VRP) class. However, many applications are out of this category, for example, robot exploration, interactive system analysis, and website usability [5].

Giacinto Angelo and Luca Grilli have contributed equally to this work.

✉ Giacinto Angelo Sgarro
giacintoangelo.sgarro@unifg.it

Luca Grilli
luca.grilli@unifg.it

[1] Department of Economics, Management and Territory (DEMeT), University of Foggia, Via A. Da Zara, 11, 71121 Foggia, FG, Italy

Considering graph theory, given a connected undirected graph $G(N, E)$ with N nodes and E edges with non-negative weights, which can be seen, for example, as the modeling of a road network where nodes are the crossroads and edges the roads, the CPP problem consists in finding a minimum-length path that starts at any node and traverses the network edges at least once before returning to the starting point [3].

One of the methodologies that can be used to solve this problem is to exploit the concept that a connected undirected graph $G(N, E)$ has at least one circuit that traverses every edge exactly once if and only if G contains exactly zero nodes of odd-degrees, i.e., if every node has an even degree, i.e., if it is an Euler graph [6].

A connected undirected graph $G(N, E)$ always has a pair number of odd-degree nodes, so it can be turned into an augmented graph $\bar{G}(N, E \cup E')$ by adding duplicated edges that transform $G(N, E)$ into an Euler graph [6].

In the CPP literature, several works use the Euler graph methodology for solving efficiently the CPP on directed and undirected graphs [3, 6–12]. However, among them, very few are the methods that belong to the class of meta-heuristics, and within this group, we can mention Genetic Algorithms (GA) [5] and DNA computing [13].

Meta-heuristics are a kind of algorithmic concepts that can be used to define heuristic methods applicable to a broad set of different problems, i.e., in other words, general-purpose algorithmic frameworks that can be applied to various optimization problems with relatively few modifications [4, 14–16].

Within the domain of meta-heuristic optimization techniques, there exists a diverse array of methodologies encompassing approaches like Ant Colony Optimization (ACO) [17–22], Genetic Algorithms (GA) [17, 19–24], Particle Swarm Optimization (PSO) [17, 18, 22, 23, 25, 26], Simulated Annealing (SA) [24, 26], Artificial Bee Colony (ABC) [18, 25], Tabu Search (TS) [19, 24], Bat Algorithm (BA) [26], Cuckoo Search (CS) [25], Firefly Algorithm (FA) [25], Grey Wolf Optimization (GWO) [25], Harmony search (HS) [25], Whale Optimization (WOA) [27], Brain Storm Optimization (BSO) [17], among others (Table 1).

Although, over the years, new meta-heuristics have been proposed in the literature, ACO, GA, and PSO have consistently demonstrated their remarkable adaptability to diverse problem structures. Consequently, these three methods find frequent application in many practical optimization scenarios (Table 1).

Among these three meta-heuristics, ACO appears to be better suited to handle the CPP problem than the others. This preference stems from its extensive history of successful applications closely aligned with the distinctive characteristics inherent to CPP. Notably, ACO's inaugural deployment transpired in the Traveling Salesman Problem (TSP) context, and subsequently, it was used successfully for a wide array of vehicle routing problems (VRP). Furthermore, it has been harnessed to tackle over one hundred problems classified within the NP-hard category. Comprehensive analysis reveals that ACO possesses two pivotal strengths: remarkable efficiency attributed to its adequate local search capabilities and the capacity to facilitate a robust probabilistic and adaptive solution generation process across the search space [4].

ACO is part of the family of meta-heuristic algorithms. It is a Swarm Intelligence (SI) method that takes inspiration from the behavior of some ant species depositing pheromones on the ground to mark favorable paths that can be followed by other members of the colony from the nest to a source of food [4].

ACO algorithms have been generally applied for solving optimization problems, generating several artificial ants that build paths that converge to exact or approximate solutions by exchanging quality information via a communication scheme inspired by ant colony working characteristics [4].

In literature, ACO was used successfully to tackle VRP problems such as shortest path problem (SPP) [28, 29] and traveling salesman problem (TSP) [4], but, to the best of our knowledge, no ACO applications are present for solving CPP.

This paper tries to fill this gap by presenting a new algorithm for CPP called ACO-CPP; the algorithm has been tested and compared with a Genetic Algorithm (GA) and a recursive algorithm that explores all possible solutions. The results show ACO-CPP proves to be efficient and consistent. Moreover, it performs equally or better than GAs.

The present work is structured as follows: Sect. 2 presents problem definitions and notations about CPP; Sect. 3 shows the general ACO framework and Sect. 4 shows its real-word applications and most notable variants; Sect. 5 describes the proposed ACO-CPP algorithm; Sect. 6 presents the experimental results; Sect. 7 presents the conclusions drawn.

2 Problem definition

The following is a list of definitions useful to present the notation used and enhance the understanding of this paper (Fig. 1).

- *Undirected graph* $G(N, E) \longrightarrow$ consists of a set of N of nodes and a set E of undirected edges where each edge

Table 1 Most recent articles related to the following two search modes: "Comparisons between meta-heuristics" and "meta-heuristic approaches for solving the vehicle routing problem, Chinese postman problem, and traveling salesman problem"

Title	Year	Method								
Metaheuristic algorithms for solving the inverse kinematics of robot manipulators [25]	2022	PSO				ABC			CS	FA/GWO/HS
Solving a traveling salesman problem using meta-heuristics [27]	2022									WOA
An agglomerative greedy brain storm optimization algorithm for solving the TSP [17]	2020	PSO	ACO	GA						BSO
An enhanced swap sequence-based particle swarm optimization algorithm to solve TSP [18]	2021	PSO	ACO			ABC				
A review of the optimization algorithms on traveling salesman problem [19]	2015		ACO	GA			TS			
A critical analysis of the bat algorithm [26]	2020	PSO			SA			BA		
Optimizing functional near-infrared spectroscopy (fNIRS) channels for schizophrenic identification during a verbal fluency task using metaheuristic algorithms [23]	2022	PSO		GA						
A statistical comparison of metaheuristics for unrelated parallel machine scheduling problems with setup times [20]	2022		ACO	GA						
Heuristic methods for vehicle routing problem with the windows [24]	2000			GA	SA		TS			
A nature inspired parameter tuning approach to cascade control for hydraulically driven parallel robot platform [22]	2015	PSO	ACO	GA						
Improved ant colony genetic algorithm for solving traveling salesman problem [21]	2020		ACO	GA						

$e_u = (n_i, n_j)$ in E connects two nodes n_i and n_j in N, and has a positive weight $w(e_u)$ [2].

- *Connected undirected graph* $\longrightarrow$ represents a graph with a path for every pair of nodes $n_i, n_j \in N$, i.e., from each node n_i to every node n_j [3].
- *Path (or chain)* $\longrightarrow$ in an undirected graph G is a sequence of adjacent edges connected through a node each other [3].
- *Circuit (or cycle)* $\longrightarrow$ consists of a closed path whose initial and final nodes coincide, i.e., a path where the starting point corresponds to the ending point [3].
- *Degree of a node $d(n_i)$* $\longrightarrow$ in an undirected graph G, is the number of edges which is incident to the node [2].
- *Eulerian path (or Eulerian trail)* $\longrightarrow$ is a path that visits every edge exactly once (allowing for revisiting vertices more times).
- *Euler's Circuit (or tour, or cycle)* $\longrightarrow$ is an Eulerian path that starts and ends on the same node, i.e., a circuit traverses every edge exactly once. A connected undirected graph G where at least an Eulerian circuit exists is defined as Euler's graph.

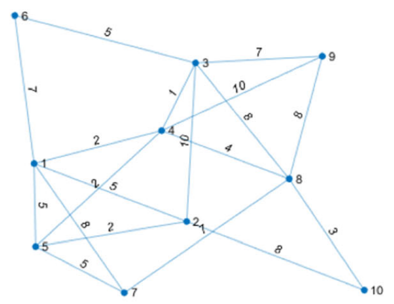
In terms of Graph Theory, in a connected undirected graph $G(N, E)$, the CPP final objective is finding a closed path (or circuit) that visits every edge at least once in such a way as to obtain a tour with the lowest total cost [5, 2].

An essential and well-known result in the literature as part of Euler's Theorem claims that a connected undirected graph G has a circuit that traverses every edge exactly once if and only if precisely zero nodes of odd-degrees are contained in it (i.e., if every node has an even degree) [3].

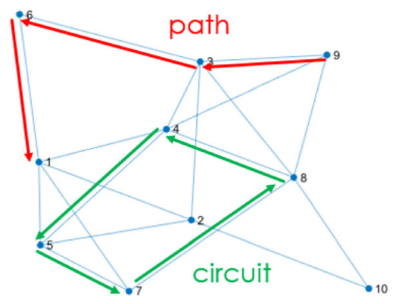
This means that when every node in a graph G has an even degree, at least one Eulerian tour exists, and all possible Eulerian tours represent optimal solutions to the CPP. Thanks to this result, a possible approach for addressing the CPP, given a graph G comprising both even and odd-degree nodes, involves the conversion of odd-degree nodes into even-degree nodes. This transformation renders the graph even, and an even graph inherently possesses Euler's tours. One possible method for solving the CPP, when provided with a graph, is to add edges $e'_u \in E'$ to G to find the minimum-cost augmentation G' of G that satisfies the even-degree property for all nodes, i.e., $d(n_i) = 2k \quad \forall n_i \in N$.

Suppose the degree of each node of a graph G is even. In that case, at least an Eulerian tour exists. Therefore, a possible CPP solution consists of finding the minimum-cost augmentation G' of G by adding edges in such a way as to satisfy the even nodes' property. In this way, an augmented Euler's graph $\bar{G} = G \cup G'$ will be generated, and the solution will be finally found by identifying a route over it [5, 2]. The problem of adding edges in such a way to satisfy

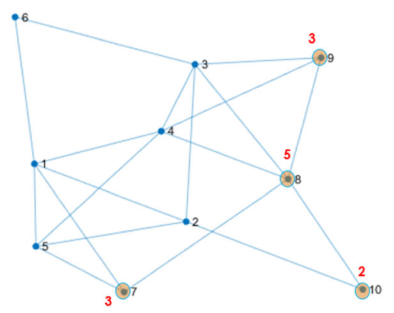
Fig. 1 Graphs representation and its characteristics



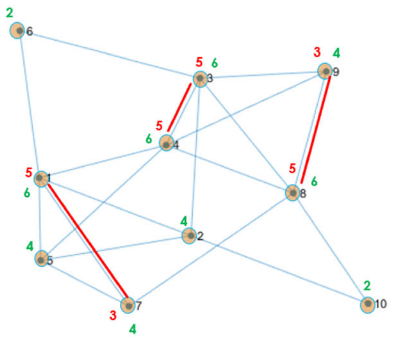
(a) Example of an undirected connected graph with indicated edge weights.



(b) The same graph with an example of a highlighted path (in red) and circuit (in green).



(c) Graph with the odd nodes highlighted.



(d) Graph with a possible augmentation.

the even nodes property can be mathematically formalized as follows:

Find the set:

$$E' \subset E \tag{1}$$

Minimize:

$$W = \sum w(e'_u), \quad e'_u \in E' \tag{2}$$

Subject to:

$$\forall n_i \in N \; \exists k \in \mathbb{N} \mid d(n_i) = 2k \tag{3}$$

$\bar{G} = G \cup G' = (N, E \cup E')$ is the Euler's graph where there is the solution of the CPP, G' is a minimum-cost augmentation of the graph, and $d(n_i)$ is the degree of the generic node n_i. In this case, it is required that each node becomes even [2].

The Handshaking Lemma states that in every finite undirected graph, the number of nodes with odd degrees is always even [30–32]. Therefore, a widespread method used to satisfy the even nodes property consists of connecting pairings of odd nodes through shortest paths (SPs). Solving the CPP, therefore, means finding the minimum-cost augmentation composed of the sum of SPs between couples of nodes with odd degrees.

As long as a method of solution for the CPP consists of finding a minimum-cost augmentation, an algorithmic method that wants to explore all the possible solutions to turn a non-Euler graph G into an Euler graph $\bar{G}$ and find in it an Euler tour would be written in the following steps:

1. Find all the odd nodes in the graph G. The total number is always even [3, 5].
2. Determine each possible pairing of odd nodes and find all the possible pairwise matching of SPs [3].
3. Explore all the possible combinations of pairings of odd node couples.
4. Select the one with the shortest total length.
5. Build a graph $\bar{G} = G \cup G'$.
6. Find an Euler tour on $\bar{G}$.

The final tour is a circuit that is a CPP optimal solution on the original graph, and its length is equal to the total length of the edges in E plus the total length of the edges E' due to the minimum-length matching [3]. An Euler graph makes it easy to find the Euler tour using well-known algorithms such as Fleury's or Hierholzer's algorithm (description in Sect. 5)[33, 3].

Exploring all the possible combinations of pairings of odd node couples is an exact method. However, it is unfeasible for problems with significant amounts of odd nodes as, given the number of odd nodes N_{odds} of a graph G, the number of possible combinations of pairings of odd node couples c is the semi-factorial of the number $n^* = N_{odds} - 1$ which corresponds to the product of the number n^* with all the odd integers preceding it, that is:

$$\begin{aligned} c &= n^*!! = n^* \cdot (n^* - 2) \cdot (\ldots) \cdot 3 \cdot 1 \\ &= \prod_{m=0}^{\frac{n^*-1}{2}} 2m + 1, \quad n^* = 2k + 1, \quad k \in \mathbb{N} \end{aligned} \tag{4}$$

3 Evolving the solution: the ACO strategy

In numerous ant species, ants release a substance known as pheromone onto the ground as they travel to and from a food source. This pheromone presence is detected by fellow ants, who then choose routes with higher concentrations. This mechanism enables ants to transport food to their nest efficiently.

Ant colony optimization is a meta-heuristic iterative algorithm formalized for the first time by Dorigo et al. [34]. Within the framework of ACO, artificial ants work collectively to construct solutions for an optimization problem. They share information about solution quality using a communication scheme (pheromone updating) that resembles the natural communication methods employed by real ants.

The objective of the pheromone update process is to enhance the pheromone levels associated with favorable or promising solutions while reducing those connected with unfavorable ones. Typically, this is accomplished through two main steps: first, a reduction in all pheromone values through pheromone evaporation, and second, an increase in the pheromone levels related to a selected group of advantageous solutions [4].

This algorithm framework generates several artificial ants at each iteration, individually building a solution by walking from node to node along a graph that encodes the problem [4]. At the moment of each choice, an ant selects one node rather than another according to a stochastic mechanism biased by a parameter τ called pheromone, a variable associated with each edge that can be read and modified by ants and whose aim is to be higher where the solution results more convenient.

More specifically, let us consider a generic node n_i connected to a certain number of edges in which an artificial ant is located; each edge has a probability of being crossed that is a combination of a stochastic mechanism value with a proportional-to-pheromone value associated with it.

At the end of each iteration, based on the quality of the solutions constructed by the ants, pheromone values are updated of an amount $\Delta\tau$ in such a way as to bias the next iteration ants in constructing solutions similar to the best ones obtained in the previous iteration. In other words, after the pheromone is deposited, subsequent artificial ants are brought to use pheromone information as a guide toward best quality search space regions [4].

Inside the iteration, the ACO meta-heuristic algorithmic general framework can be traced back to three main phases as shown in Algorithm 1:

1. *AntsGeneration:* a set of N_{ant} artificial ants is generated and constructs solutions traveling through feasible steps without violating the constraints. Given a graph G, the process can be imagined as a walk on it where the choice of each edge is guided by a stochastic mechanism biased by the pheromone, respectively, associated. The rule of the stochastic-pheromone choice varies across the different ACO algorithms proposed in literature [4].
2. *LocalSearch:* included in state-of-the-art ACO algorithms [4] consists of improving the solutions obtained through a local search.
3. *PheromoneUpdating:* the pheromone is updated according to pheromone values that are associated with good quality artificial ant solutions, and it is decreased through pheromone evaporation [4].

Algorithm 1 ACO general framework [4]

```
Set parameters, initialize pheromone
while termination condition not fulfilled do
    AntsGeneration
    LocalSearch (optional)
    PheromoneUpdating
end while
```

4 ACO: real-word applications and most notable variants

ACO has widely applied to various real-world scenarios. Throughout its history, the typology of real-life applications where ACO has found greater diffusion are routing, assignment, scheduling, and subsetting problems. However, it is essential to note that this list is not exhaustive, as ACO and its variants have demonstrated their effectiveness in an extensive range of applications [4]. Recent examples of ACO applications include its utilization in solving the Waste Collection Routing Problem (WCRP). This problem involves designing an optimal route to serve all customers (represented as nodes) with minimal total travel time or distance while employing the fewest number of vehicles, considering specific constraints such as vehicle capacity [35]. ACO has also been employed to address the Dynamic Traveling Salesman Problem (DTSP). In this scenario, a supplier must deliver parcels to customers, with parcels arriving at the depot during distribution. The time a parcel arrives at the depot is referred to as its release date [36]. In E-logistics, ACO has been instrumental in tackling the VRP for service providers, where vehicles must strategically visit pick-up nodes, such as warehouses, before making customer deliveries. This multifaceted problem entails minimizing total travel costs while accommodating real-world constraints like heterogeneous vehicles, capacity limits, time windows, and driver working durations [37]. Additionally, ACO has been effectively employed in optimizing trajectory tracking for nonlinear three-rigid-link maneuvers (RLM) [38] and minimizing non-productive tool travel time in drilling processes [39]. ACO's versatility also extends into the realm of optimizing neural network algorithms [40]. In the context of cloud computing, where service providers grapple with resource allocation, security, privacy, and virtual machine migration, ACO has proven successful in scheduling heterogeneous tasks, thus enhancing Quality of Service (QoS) for clients as the volume of client requests escalates within cloud environments [41]. Lastly, ACO has recently been employed to generate optimal trajectories for industrial robots in machining and additive manufacturing applications [42]. In summary, ACO's versatility and effectiveness have led to its adoption in various applications, showcasing its adaptability and value in addressing complex real-world problems.

Numerous ACO algorithms have been introduced in the academic literature. In this context, we outline the primary characteristics of the three most prominent variants, namely Ant System (AS), MAX-MIN Ant System (MMAS), and Ant Colony System (ACS).

- **Ant System (AS)**: AS is primarily characterized by the simultaneous pheromone value updates performed by all k ants that have constructed a solution during each iteration [43]. In this scenario, the pheromone update process is executed as follows:

$$\tau(i,j) \leftarrow (1-\rho)\tau(i,j) + \sum_{k=1}^{Nant} \Delta\tau_k(i,j) \tag{5}$$

  where ρ represents the evaporation rate, N_{ant} denotes the total number of ants, and $\Delta\tau_k(i,j)$ is the amount of pheromone deposited on the edge (i, j) by ant k. Ant System is the first ACO algorithm proposed in the literature [43] and is generally used for solving routing problems.
- **MAX-MIN Ant System (MMAS)**: distinctive features of MMAS include the exclusive pheromone trail updates performed solely by the best ant and the imposition of a pheromone value boundary [44]. The pheromone update process is executed in the following manner:

$$\tau(i,j) \leftarrow [(1-\rho)\tau(i,j) + \Delta\tau_{\text{Best}}(i,j)]_{\tau_{min}}^{\tau max} \tag{6}$$

  here, $\tau_{\max}$ and τ_{min} represent the upper and lower bounds set for pheromone levels, often determined empirically. The operator $[x]_{\tau_{min}}^{\tau max}$ is defined as follows:

$$[x]_{\tau_{min}}^{\tau max} = \begin{cases} \tau_{\max} & x > \tau_{\max} \\ \tau_{\max} & x < \tau_{\min} \\ x & \text{otherwise} \end{cases} \tag{7}$$

  and $\Delta\tau_{\text{Best}}(i,j)$ is as follows:

$$\Delta\tau_{\text{Best}}(i,j) = \begin{cases} 1/L_{\text{Best}} & e_u \in \text{Tour} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

  where L_{Best} represents the length of the tour undertaken by the best ant. The determination of the best tour length depends on the choices made by the designer, whether it is based on the best ant of the current iteration (iteration-best) or the best ant observed since the beginning of the algorithm (best-so-far).
- Ant Colony System (ACS): the most interesting contributions of ACS [45, 46] is the introduction of a *local pheromone update* in addition to the pheromone update performed at the end of the construction process (*offline pheromone update*). The local pheromone update is performed by all the ants after each construction step. Each ant applies it only to the last edge traversed by using:

$$\tau(i,j) = (1-\phi)\tau(i,j) + \phi\tau_0 \tag{9}$$

  where $\phi \in (0,1]$ is the pheromone decay coefficient, and τ_0 is the initial value of the pheromone. The primary aim of the local update is to introduce diversity into the search conducted by subsequent ants within an iteration. By decreasing the pheromone concentration on traversed edges, ants encourage the selection of different edges by subsequent ants, generating distinct solutions. This reduces the likelihood of multiple ants producing identical solutions during a single iteration.

5 ACO-CPP: ant colony optimization for CPP

5.1 General algorithm framework

In the context of ACO, an algorithm for solving the CPP can be conceptualized as follows: The CPP involves identifying the optimal pairings of odd nodes through SPs to minimize the total cost. In our ACO-CPP approach, each ant consists of a set of pairings of odd nodes, where each pair represents an SP. The quality of an ant's path is determined by the total length, or cost, which corresponds to the sum of its SPs.

Over several iterations, a group of ants is generated. During the initial iteration, ants are created using a heuristic scheme. Specifically, ants select random odd nodes to move between, following a rule prioritizing higher likelihood pairings resulting in shorter SPs. This initial iteration serves as a heuristic exploration phase to find better solutions. At the end of each iteration, ants that have discovered superior solutions deposit a significant amount of pheromone along their paths, while those with inferior solutions contribute less. Throughout the paths marked by pheromones, the evaporation chemical agent works to decrease pheromone levels across the solution space uniformly.

Starting from the second iteration, the pheromone parameter exerts influence. During the construction of each path, the ant is influenced by the heuristic factor and the pheromone level, which represents the collective experience of previous ants in the search. As iterations progress, the pheromone component increasingly affects the choices of the ants, leading to more ants following a common path, which indicates algorithm convergence toward a solution (Fig. 2). If the search space has been sufficiently well explored and the pheromone has guided ants to converge along the best path, the global optimum of the CPP can be attained.

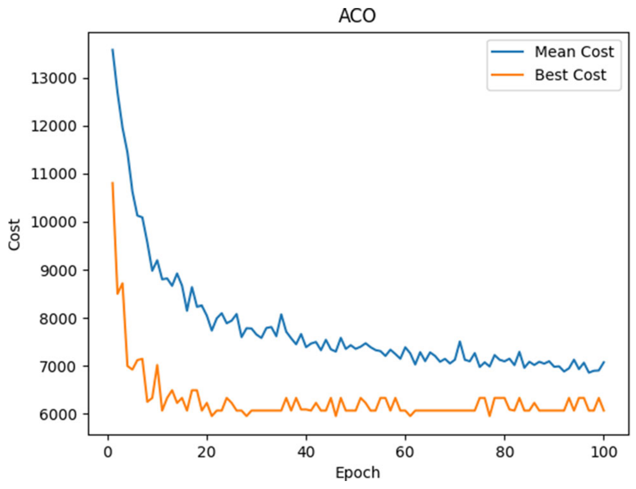


Fig. 2 Example of a convergence graph obtained by using ACO-CPP with 50 ants for 100 iterations on a graph with 48 nodes, 24 of which have odd degrees, and with edge weights ranging from 1 to 1000

In light of this operational framework, we introduce the Ant Colony Optimization for CPP (ACO-CPP). The algorithm begins by constructing a matrix denoted as D, representing the SPs between pairs of odd vertices within the initial graph G. Subsequently, the ants aim to identify the optimal combination of SPs whose total length yields the minimum-cost augmentation, denoted as G'. This entails minimizing $\sum w(e_u)'$. Finally, Hierholzer's algorithm obtains the Eulerian graph $\bar{G}$ by finding a possible Euler tour. The general ACO-CPP framework is shown in Fig. 3.

More in detail, the ACO-CPP algorithm starts from the undirected graph $G = (N, E)$ and gives as output an undirected graph $\bar{G} = (N, E \cup E')$ running the following procedure:

1. ***Get odd nodes:*** All the N_{odds} odd-degree nodes are identified in G; N_{odds} is always even [3].
2. ***Matrix of odd-pairs-SPs construction:*** a square anti-diagonal matrix of distances D of order N_{odds} is built, i.e., a matrix which contains in its cells the SPs between odd nodes. Each index represents an odd node, and each cell is the SP cost between the corresponding odd-degree nodes, i.e., the minimum-length matching. In other words, in such a matrix, for each pairing, we find the edges that connect the odd vertices with the shortest possible path. Dijkstra's algorithm calculates the shortest route between each pair.
3. ***ACO search:*** The core of the ACO-CPP algorithm presented in this paper consists of finding the best combination of SPs between odd nodes (pairwise matching), which minimizes the total cost of the augmentation through an ACO search. According to this, two matrices, η and τ, are generated starting from the matrix D described as follow:
   - ***Attractiveness matrix***(η): matrix of order N_{odds} computed as the reciprocal of D changing the diagonal values as 0 as in Equation (10). This matrix represents the contribution, during the selection of the shortest path (SP) by an ant, of choosing one SP over another with a higher probability based on the length of the SP:

     $$\eta(i,j) = \frac{1}{D(i,j)}, \quad i \neq j; \; \eta(i,i) = 0. \tag{10}$$

   - ***Pheromone matrix***(τ): matrix of order N_{odds} that at the starting point has for each cell the user-selected value τ_0 as in Equation (11). As the iterations progress, this matrix will see its values change due

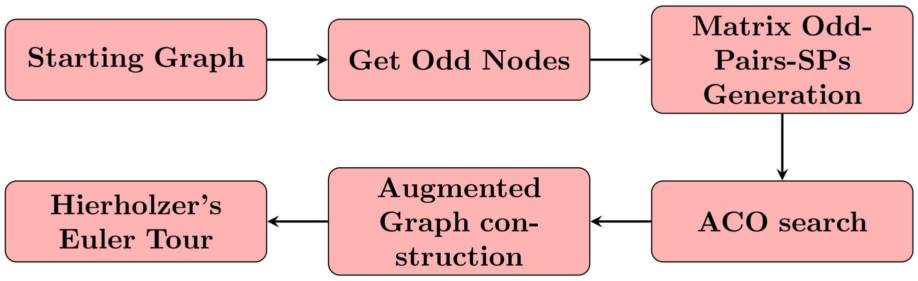


Fig. 3 The ACO-CPP general framework

to the contributions of the ants from past iterations. Specifically, its cell values will change positively the more past ants have found particular SPs more convenient, considering their entire path. Simultaneously, all the cell values will change negatively due to the contribution of pheromone evaporation. This matrix represents the contribution, during an ant's selection of an SP, of choosing one SP over another with a higher probability based on what the experience of the past ants suggests as being more convenient.

$$\tau = \begin{pmatrix} \tau_0 & \dots & \tau_0 \\ \vdots & \ddots & \vdots \\ \tau_0 & \dots & \tau_0 \end{pmatrix} \tag{11}$$

At this point, it is necessary to find the best combination of odd pairings whose SPs sum is the least possible. Each combination represents an ant and signifies the additional edges needed to solve CPP.

The ants generation consists in the generation of N_{ant} ants `for` $it_{\max}$ iterations. Each ant is associated with a probability matrix P, defined as a Hadamard product (element-wise product) between η and τ. At the beginning of the ants construction, this matrix appears the same for all ants in the current iteration. However, its values are subsequently modified differently based on the choices made by the ants during their construction process. Each cell represents a potential choice (SP) made by an ant as it traverses its path. Following this scheme, when an ant selects one cell over another, it implies that the ant chooses an SP between two nodes rather than an alternative one. Specifically, matrix P is described as follows:

- ***Probability matrix*** (P): matrix defined as the Hadamard product of Equation (12) with α and β, two parameters user-selected called *pheromone* and *heuristic weights*, which aim to control the relative importance of the pheromone versus the heuristic information [4]:

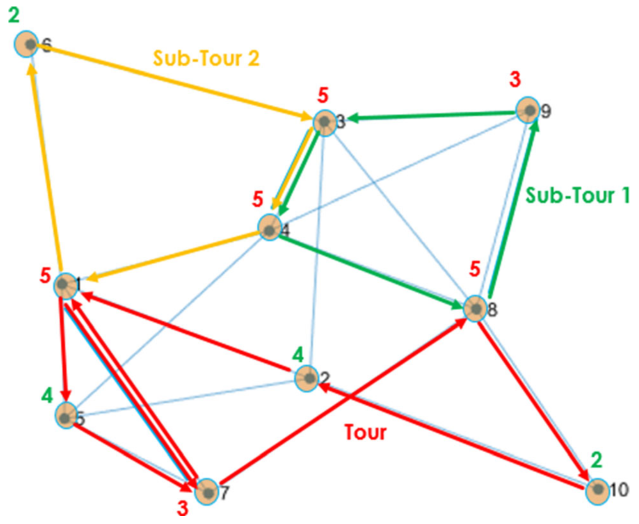


Fig. 4 Example of constructing a Hierholzer's Euler Tour using the previously augmented graph as an illustration

$$P = (\tau\alpha) \circ (\eta\beta). \tag{12}$$

Such a matrix has the aim to outline the pairings `for` $N_{odds}/2$ times. At first, a normalization of matrix P is generated:

$$P(i,j) \leftarrow \frac{P(i,j)}{\sum_{i,j} P(i,j)}, \tag{13}$$

then, through a selection criterion, a couple of odd nodes are selected; lastly, all the elements present in the rows and columns corresponding to the chosen nodes are set to 0. These three steps are repeated `for` $N_{odds}/2$ times. Our ACO-CPP used the *Roulette selection* method as a selection criterion. *Roulette selection* is a kind of selection in which the probability that one solution is selected is directly proportional to the pheromone value owned by the solution itself [47]. To establish a connection between this definition and the current problem, let us consider a scenario in which we have a probability matrix. From this matrix, we must select a single cell representing a potential SP between two odd nodes. This process involves two main steps.

Firstly, we normalize all the values within the matrix by dividing them by the sum of all the values in matrix P. This normalization ensures that the values are appropriately scaled. Secondly, we compute the cumulative sum of all the elements in the matrix, resulting in a set of values falling within the range [0, 1]. At this stage, a random number in the range [0, 1] is generated, and the cumulative sum determines the boundaries within which this number falls. These boundaries indicate the selected indexes corresponding to the choice.

After the construction of ants during an iteration, a pheromone update is applied, which has two components:

- ***Pheromone reinforcement:*** `for` each ant, pheromone is updated thought the formula of Equation (14), where $\Delta\tau(i,j)$ is the quantity of pheromone laid on the edge (n_i, n_j) by each ant, L is the length of the tour constructed, and Q is a constant user-selected:

$$\tau(i,j) \leftarrow \tau(i,j) + \Delta\tau(i,j), \quad with \quad \Delta\tau(i,j) = \frac{Q}{L}, \tag{14}$$

- ***Pheromone evaporation:*** `for` each iteration, pheromone evaporates thought the formula of Equation (15) where ρ is a constant user-selected called Evaporation Rate:

$$\tau(i,j) \leftarrow (1-\rho)\tau(i,j), \tag{15}$$

Similarly to many other ACO algorithms in the literature [4], the ACO-CPP algorithm has features such as the pheromone updating in both its forms pheromone (reinforcement and evaporation) and the probability matrix. However, the latter is used as a matrix of SPs between odd nodes and presents the consequent manipulations described in "Probability matrix (P)" point to adapt it to the CPP structure problem. The graphic representation of the ACO search is shown in Fig. 5.

4. ***Augmented Graph construction:*** an *Ant* is a possible solution to the CPP problem (combination of odd pairings), and the *WinnerAnts* are the best solutions the ACO-CPP algorithm has found. The best-found combination of extra edges G' is used to generate a new augmented graph $\bar{G} = G \cup G' = (N, E \cup E')$ that contains no nodes of odd-degree; such a graph, known as Euler Graph, has the property of containing at least one possible route that may be a solution for the CPP [2], and, such a possible route, known as Euler Tour, can be found through algorithms like Hierholzer's one [33, 3].
5. ***Hierholzer Euler Tour:*** An Euler tour is found using Hierholzer's algorithm. The general framework of Hierholzer's algorithm is the following: choose any starting node n_i, and follow a trail of edges from that node until returning to n_i. It is impossible to get stuck at any node other than n_i, because the even degree of all vertices ensures that, when the trail enters another node n_j, an unused edge must be left n_j. The tour formed this way is a closed tour but may not cover all the vertices and edges of the initial graph. As long as there exists a node n_k that belongs to the current tour but that has adjacent edges not part of the tour, start another trail from n_k, following unused edges until returning to n_k, and join the tour formed in this way to the previous tour. Since we assume the original graph is connected, repeating the previous step will exhaust all edges of the graph [48]. Such a circuit represents a CPP optimal solution whose length is equal to the total length of the edges in G plus the total length of the edges in the minimum-length matching [3].

5.2 Encoding

ACO-CPP algorithm is written using PYTHON, and graphs are represented using a weighted adjacency matrix and node table encoding. In this context, the graph is represented as a square weighted matrix which has the names of the nodes of the graph as row and column indexes so that, if there exists an edge from node n_i to node n_j, in the place (n_i, n_j) there is a number corresponding to its weight. Otherwise, it is 0. The weighted node table encoding is a matrix N_{edges}x3 where the three columns indicate the starting node, the ending node, and the weight of an edge of G.

Weighted adjacency matrix encoding was beneficial in getting the odd nodes from the graph, and weighted node table encoding showed its advantage for the random generation of weights during the graph construction.

We used the PYTHON object class for artificial ants representation, characterized by two properties: *Tour* and *Cost*. The *Tour* is a list of lists where each sub-list is an odd vertices pairing. *Cost* is a float64 variable that contains the SPs sum of all the odd pairings. The ant object structure is defined as in Algorithm 2.

Algorithm 2 Ant encoding

```
$class Ant:$
$def(self, Tour, Cost):$
$self.Tour = Tour = [[n_{start}, n_i], [n_j, n_k], \dots, [n_t, n_{end}]]$
$self.Cost = Cost = \sum_{\frac{N_{odds}}{2}} SPs = \sum_{\frac{N_{odds}}{2}} D(i,j)$
```

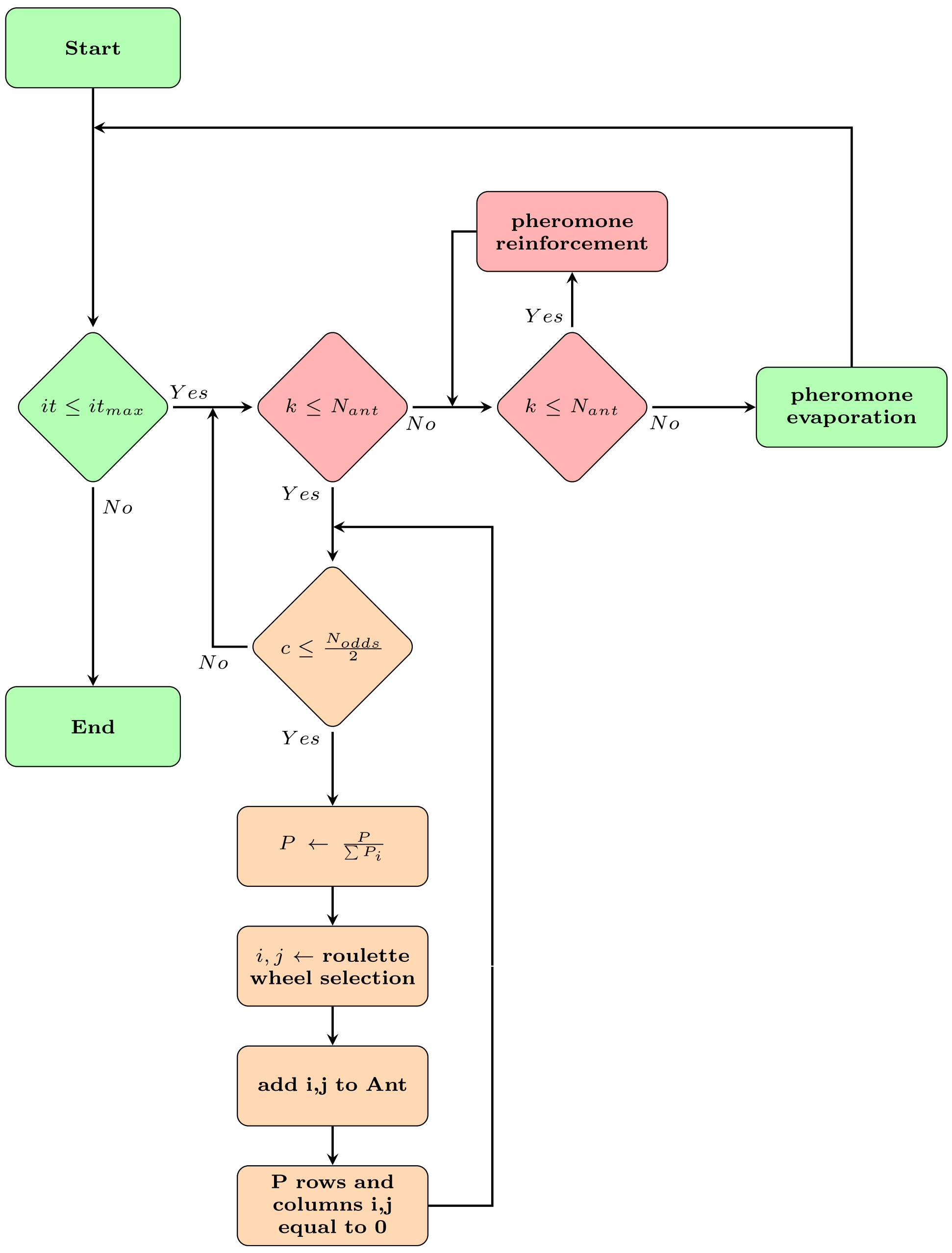


Fig. 5 Workflow of the core of the ACO-CPP. Red boxes represent the nested `for` iterations inside the main `for` cycle (green boxes). Besides, in one `for` cycle, there is a sub-nesting `for` cycle (orange boxes)

6 Experimental analysis

To evaluate the ACO-CPP performances, we developed an algorithm to execute various test cases. In each test case, we create a connected graph and then identify the best solution using different combinations of:

- Recursive algorithm proposed by Araz Sharma[1]
- Genetic Algorithms of Jiang Hua and Kang Li-shan[5].
- ACO-CPP algorithm.

The following subsections will describe the various experiment components, the experimental setup and the experiment results.

6.1 Graph generation

Our graph generation algorithm can generate a connected undirected graph with a desired number of nodes that are randomly connected and with each weight value ranging within the integer values [1; 1000]. Such a function was designed to manage the number of nodes and the count of

[1] Sharma, A., Chinese Postman in Python: Detailed Implementation & Explanation, with particular emphasis on a unique approach to generating Odd Vertex Pairings using Recursion. Towards Data Science, (2020), available at: https://towardsdatascience.com/chinese-postman-in-python-8b1187a3e5a.

odd nodes. Specifically, in the first phase, an undirected graph with random connections where it is guaranteed the property of connection G_{FC} is generated by considering as an object the number of nodes provided (N_{nodes}). Then, considering the number of odd nodes (N_{odds}) user-required, an augmented graph G_A is built by randomly connecting the nodes. The edges of both graphs are considered, and if there are present self-loops or extra-edges, they are deleted. New edges are added until the number of odd nodes user-desired is obtained. Lastly, the two graphs are merged, getting G, a not-weighed graph, and as the last step, each edge is assigned a random weight ranging from 1 to $W\text{max} = 1000$. The pseudo-code of graph construction is provided in Algorithm 3.

Algorithm 3 Undirected connected graph generation pseudo-code

```
Input: $N_{nodes}, N_{odds}, W_{max}$
Output: $G_{FC}, G_A, G_{weighted}$

$G_{FC} \leftarrow generate\_fc(N_{nodes})$

while $N_{odds}$ not fulfilled do
    $G_A \leftarrow generate\_edge$

    if $SelfLoops$ or $Duplicates$ then
        $G_A \leftarrow delete(SelfLoops$ or $Duplicates))$
    end if
end while

$G \leftarrow merge(G_{FC}, G_A)$
$G_{weighted} \leftarrow assign\_weight(G, W_{max})$
```

6.2 Recursive algorithm

In the recursive algorithm, starting from the identified odd nodes, all possible combinations of pairings are computed, along with the corresponding subsequent cost of graph augmentation. Among all the costs, the lowest value is selected and stored.

6.3 Genetic algorithm

Genetic Algorithms have an object coding identical to ACO-CPP. There is a matrix of SPs, and solutions are coded as *Genotypes*, which are objects made by a list of pairing of odd nodes (*Tour*) and a relative cost graph augmentation (*Cost*).

In such an algorithm, an initial population is considered `for` a certain number of iterations. Then, using selection criteria, a portion p_m and p_c of it is chosen to produce a new generation of child solutions through genetic operators like *mutation* and *crossover*. Finally, from all the solutions in the current iteration, a subset is selected to form the next generation for the subsequent iteration. *Roulette wheel selection* is always applied as a selection criterion, and the best solution is selected at the end of the algorithm.

During the *mutation* process, for each $p_{m,i}$ *genotype*, two random integers are generated within the range $[1, N_{odds}]$, which will represent the indices in the Genotype Tour to be exchanged. To provide an example, let us assume $N_{odds} = 8$. When the mutation operator is applied to a genotype, it randomly generates two indices, for instance, $i = 3$ and $j = 7$. The operator then exchanges the genotype values at positions i and j. As a result:

$$Gentype(i)_{new} = Genotype(j)_{old} \quad Gentype(j)_{new} = Genotype(i)_{old} \tag{16}$$

During *crossover*, a pair of *genotypes* is selected. A pair of indexes corresponding to the position of a pair of odd nodes in *genotypes* tour is individuated. Odd node values are exchanged along those positions, and consequently, the remainder of the *genotype* list tours are adjusted. To provide an example, let us assume $N_{odds} = 8$. When the crossover operator is applied to a genotype, it randomly generates two adjacent indices, denoted as i and j, both starting from an odd number. For instance, let us consider $i = 3$ and $j = 4$. Next, two genotypes, labeled a and b, are randomly selected from the population, and the odd nodes between them are exchanged based on the specified indices, with adjustments made to the remaining elements. This process can be described as follows:

$$\begin{aligned} Gentype(i)_{a,new} &= Genotype(i)_{b,old} \\ Gentype(j)_{a,new} &= Genotype(j)_{b,old} \end{aligned} \tag{17}$$

$$\begin{aligned} Gentype(i)_{b,new} &= Genotype(i)_{a,old} \\ Gentype(j)_{b,new} &= Genotype(j)_{a,old} \end{aligned} \tag{18}$$

6.4 ACO-CPP algorithm

The ACO-CPP algorithm starts from the odd nodes information, generates the matrix of SPs D, and then `for` $it\text{max}$ iterations develop $N\text{ant}$. Each ant is built in this way: at the beginning are constructed the initial pheromone matrix τ and the heuristic matrix η, then the matrix $P = (\tau\alpha) \circ (\eta\beta)$. Such a matrix `for` the number of odd pairings divided 2 times, `for` $N\text{ant}$ ants, in a first moment, is updated in the matrix $P(i,j) = \frac{P(i,j)}{\sum P(i,j)}$, then, using the function *Roulette wheel selection*, finds the two odd nodes to connect and sets all the elements of both rows and columns of the odd nodes indexes to 0.

After each iteration, pheromone updating is applied as illustrated in Equation (14) and Equation (15). Specifically, at the end of each iteration, every ant contributes to updating the matrix τ. This contribution, the pheromone reinforcement, is more significant the shorter the total path the ant has traversed along the cells of τ, which corresponds to the tour undertaken by the ant itself (Equation (14)). Following the reinforcement phase, the evaporation phase is implemented, involving a uniform reduction of all τ values by an amount specified in Equation (15). The underlying idea is to utilize pheromone reinforcement to emphasize more favorable paths while employing pheromone evaporation to diminish less favorable paths.

The ACO-CPP algorithm is built in such a way as to store the artificial ant, which shows the best solution of all (*Winner Ant*). During the experiment, its hyperparameter settings are those of Algorithm 4.

Algorithm 4 ACO-CPP parameters set in the experiment

Algorithm 4 ACO-CPP parameters set in the experiment

```
$N_{ant} := [10, 20, 30, 40]$                 ▷ Number of Ants (Population Size)
$it_{max} := [10, 20, 30, 40]$                ▷ Maximum number of iterations
$N_{odds} := [4, 6, 8, 10, 12]$               ▷ Number of Odd Nodes
$Q := 1$                                      ▷ Constant
$\tau_0 := 10\frac{Q}{N_{odds}\bar{D}}$       ▷ Initial pheromone
$\alpha := [0.5, 1, 1.5]$                     ▷ Phromone Exponential Weight
$\beta := [0.5, 1, 1.5]$                      ▷ Heuristic Exponential Weight
$\rho := 0.05$                                ▷ Evaporation Rate
```

6.5 Experimental setup

To evaluate the ACO-CPP performance, three different test cases were carried out. We conducted the experiments 300 times to measure the effectiveness and the consistency of ACO-CPP, setting the ACO-CPP parameters as in Algorithm 4. For both meta-heuristics, we varied the choice of the number of solutions explored (ants for ACO, and p_m and p_c for GA) and the number of iterations within the range of 10 to 40. A more comprehensive description of the experiment is as follows:

- **Experiment 1:** Recursive-GA-ACO comparison conducted to measure the meta-heuristics ability to obtain the global optimum (Algorithm 5). We considered graphs with varying numbers of odd nodes (ranging from 4 to 12) and various combinations of hyperparameters designed to maintain a similar number of solutions explored by the meta-heuristics. We performed 300 measurements for different combinations of graphs and hyperparameters to assess the augmentation brought about by ACO and GA. This allowed us to determine how often GA and ACO achieved the global optimum during their respective runs.
- **Experiment 2:** GA-ACO statistics comparison for measuring consistency and reproducibility (Algorithm 6). Five graphs with varying numbers of odd nodes (ranging from 4 to 12) are generated (see Fig. 6 in Sect. 1). Using different combinations of hyperparameters designed to maintain a similar number of solutions explored by the meta-heuristics, we conducted 300 measurements for each graph and hyperparameter combination to calculate the graph augmentation produced by ACO and GA. From these measurements, we computed the average and standard deviation. The lower the average and standard deviation, the more consistent the algorithm can find the best solution.
- **Experiment 3:** Recursive-GA-ACO comparison conducted to measure the ability of meta-heuristics to achieve the global optimum when varying the ACO parameters α and β (Algorithm 7). This test case is identical to the first one but explores different combinations of hyperparameters α and β. The aim was to investigate whether the proposed experimental setup inherently exhibits characteristics that result in different modes of exploring the search space to find the best solution, depending on the chosen graph parameters. The goal was to find the best combination of hyperparameters for solving it most effectively.

To conduct the experiments, we employed a computer equipped with 12 GB of RAM and an i5 processor (2.30 GHz). With these specifications, the computer could solve problems using the recursive algorithm and store all possible solutions in memory for graphs with up to 12 odd nodes within a relatively short time frame. It required several minutes to solve problems with 14 nodes, but it could not handle problems with 16 nodes or more. This description emphasizes as in this case, it is possible to find exact solutions for the CPP, but in general, as problems become more complex, finding exact solutions becomes impractical or exceedingly time-consuming, and the meta-heuristics advantages start to show up.

The primary challenge in meta-heuristics is to obtain the global optimum without getting trapped in a local minimum. To assess the ability of ACO-CPP to attain the global optimum with a certain level of accuracy, it is necessary to compare it with an exact algorithm, such as the recursive algorithm. Additionally, it is crucial to determine whether the inherent working scheme of ACO-CPP is better suited for solving problems like CPP compared to other meta-heuristics, therefore requiring a comparison with GA.

Algorithm 5 Experiment 1

```
Input: $N_{odds}, N_{nodes}, MaxIt, N_{ant}, P_m, P_c$
Output: $Best_{Recursive}, Best_{GA}, Best_{ACO}$

$[N_{odds}/N_{nodes}] = [4/10 \;\; 6/12 \;\; 8/18 \;\; 10/20 \;\; 12/24]$
$MaxIt = [10 \;\; 20 \;\; 30 \;\; 40]$
$N_{ant} = [10 \;\; 20 \;\; 30 \;\; 40]$
$[P_m/P_c] = [6/6 \;\; 10/10 \;\; 16/16 \;\; 20/20]$

for all combinations of $[N_{odds}/N_{nodes}]$, $N_{ant}/[P_m/P_c]$, and $MaxIt$ do
    for 300 times do
        $G_{weighted} \leftarrow graph\_generation([N_{odds}/N_{nodes}])$
        $Best_{Recursive} \leftarrow recursive\_algoritm(N_{odds}, G_{weighted})$
        $Best_{GA} \leftarrow GA\_algorithm([P_m/P_c], MaxIt)$
        $Best_{ACO} \leftarrow ACO\_algorithm(N_{ant}, MaxIt)$
        $Counter_{GA}, Counter_{ACO} \leftarrow Counter(Best_{Recursive}, Best_{GA}, Best_{ACO})$
    end for
end for
```

Table 2 Number of tests where the best solution is found by varying the parameters P_m, P_c and $it_{\max}$ for GA, while N_{ant} and $it_{\max}$ for ACO (ACO best results over GA in bold)

Genotypes/Ants		Number of tests where the best solution is found				
$[P_m/P_c]$ N_{ant}	$it_{\max}$	4 Odds	6 Odds	8 Odds	10 Odds	12 Odds
6/6	10	300/300	299/300	**220/300**	**72/300**	**8/300**
10	10	300/300	297/300	**282/300**	**242/300**	**192/300**
6/6	20	300/300	300/300	**271/300**	**134/300**	**39/300**
10	20	300/300	296/300	**292/300**	**285/300**	**243/300**
10/10	10	300/300	300/300	**275/300**	**121/300**	**21/300**
20	10	299/300	297/300	**300/300**	**279/300**	**248/300**
10/10	20	300/300	300/300	299/300	**201/300**	**57/300**
20	20	300/300	300/300	293/300	**286/300**	**267/300**
10/10	30	300/300	300/300	300/300	**258/300**	**103/300**
20	30	300/300	297/300	295/300	**292/300**	**261/300**
16/16	20	300/300	300/300	300/300	**254/300**	**97/300**
30	20	300/300	299/300	294/300	**287/300**	**267/300**
16/16	30	300/300	300/300	300/300	**279/300**	**145/300**
30	30	300/300	300/300	298/300	**287/300**	**280/300**
16/16	40	300/300	300/300	300/300	**354/362**	**163/242**
30	40	300/300	298/300	295/300	**341/362**	**213/242**
20/20	30	300/300	300/300	300/300	291/300	**181/300**
40	30	300/300	299/300	298/300	288/300	**278/300**
20/20	40	300/300	300/300	300/300	296/300	**226/300**
40	40	300/300	300/300	296/300	291/300	**273/300**

Algorithm 6 Experiment 2

```
Input: $N_{odds}, N_{nodes}, MaxIt, N_{ant}, P_m, P_c$
Output: $Best_{GA}, Best_{ACO}$

$[N_{odds}/N_{nodes}] = [4/10\ \ 6/12\ \ 8/18\ \ 10/20\ \ 12/24]$
$MaxIt = [10\ \ 20\ \ 30\ \ 40]$
$N_{ant} = [10\ \ 20\ \ 30\ \ 40]$
$[P_m/P_c] = [6/6\ \ 10/10\ \ 16/16\ \ 20/20]$

for all combinations of $[N_{odds}/N_{nodes}]$, $N_{ant}/[P_m/P_c]$, and $MaxIt$ do
    $G_{weighted} \leftarrow Graph\_Generation([N_{odds}/N_{nodes}])$
    for 300 times do
        $Best_{GA,i} \leftarrow GA\_algorithm([P_m/P_c], MaxIt)$
        $Best_{ACO,i} \leftarrow ACO\_algorithm(N_{ant}, MaxIt)$
    end for
end for

$mean(Best_{GA,i}),\ \ std\_dev(Best_{GA,i})$
$mean(Best_{ACO,i}),\ \ std\_dev(Best_{ACO,i})$
```

Algorithm 7 Experiment 3

```
Input: $N_{odds}, N_{nodes}, MaxIt, N_{ant}, P_m, P_c$
Output: $Best_{Recursive}, Best_{GA}, Best_{ACO}$

$[N_{odds}/N_{nodes}] = [4/10\ \ 6/12\ \ 8/18\ \ 10/20\ \ 12/24]$
$MaxIt = [10\ \ 20\ \ 30\ \ 40]$
$N_{ant} = [10\ \ 20\ \ 30\ \ 40]$
$\alpha = [0.5\ \ 1\ \ 1.5]$
$\beta = [0.5\ \ 1\ \ 1.5]$
$[P_m/P_c] = [6/6\ \ 10/10\ \ 16/16\ \ 20/20]$

for all combinations of $[N_{odds}/N_{nodes}]$, $N_{ant}/[P_m/P_c]$, and $MaxIt$ do
    for 300 times do
        $G_{weighted} \leftarrow graph\_generation([N_{odds}/N_{nodes}])$
        $Best_{Recursive} \leftarrow recursive\_algoritm(N_{odds}, G_{weighted})$
        $Best_{GA} \leftarrow GA\_algorithm([P_m/P_c], MaxIt)$
        $Best_{ACO} \leftarrow ACO\_algorithm(N_{ant}, MaxIt, [\alpha/\beta]))$
        $Counter_{GA}, Counter_{ACO} \leftarrow Counter(Best_{Recursive}, Best_{GA}, Best_{ACO})$
    end for
end for
```

Table 3 Minimum, maximum, average, and standard deviation of graph augmentations when the best solution is found in 5 graphs (ranging from 4 to 12 odd nodes) over 300 runs

$it_{\max}$		Minimum, maximum, average and standard deviation of the best meta-heuristic solutions									
		4 Odds		6 Odds		8 Odds		10 Odds		12 Odds	
$[P_m/P_c]$ N_{ant}	Stat	GA	ACO	GA	ACO	GA	ACO	GA	ACO	GA	ACO
10	Min	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2315.0	2315.0
	Max	1270.0	1270.0	1197.0	1197.0	2621.0	2157.0	3217.0	1741.0	4393.0	2686.0
6/6	Average	1270.0	1270.0	1197.0	1197.0	2300.4533	2157.0	2141.2633	1741.0	3300.8933	2335.8933
10	Std Dev	0	0	0	0	181.4907	0	382.6489	0	455.6600	60.3899
20	Min	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2315.0	2315.0
	Max	1270.0	1270.0	1197.0	1197.0	2621.0	2157.0	3031.0	1741.0	4082.0	2396.0
6/6	Average	1270.0	1270.0	1197.0	1197.0	2211.7866	2157.0	1923.79	1741.0	2989.6566	2316.35
10	Std Dev	0	0	0	0	125.1705	0	286.9127	0	448.7917	10.3868
10	Min	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2315.0	2315.0
	Max	1270.0	1270.0	1197.0	1197.0	2621.0	2157.0	2505.0	1741.0	4082.0	2396.0
10/10	Average	1270.0	1270.0	1197.0	1197.0	2213.5066	2157.0	1964.7366	1741.0	3037.0233	2315.27
20	Std Dev	0	0	0	0	131.6749	0	301.2844	0	404.4056	4.6765
20	Min	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2315.0	2315.0
	Max	1270.0	1270.0	1197.0	1197.0	2481.0	2157.0	2347.0	1741.0	3685.0	2315.0
10/10	Average	1270.0	1270.0	1197.0	1197.0	2165.64	2157.0	1785.4733	1741.0	2708.5566	2315.0
20	Std Dev	0	0	0	0	52.2859	0	143.9865	0	346.6172	0
30	Min	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2315.0	2315.0
	Max	1270.0	1270.0	1197.0	1197.0	2621.0	2157.0	2453.0	1741.0	4006.0	2315.0
10/10	Average	1270.0	1270.0	1197.0	1197.0	2165.0266	2157.0	1799.2766	1741.0	2672.3366	2315.0
20	Std Dev	0	0	0	0	52.5542	0	169.1625	0	351.3382	0
20	Min	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2315.0	2315.0
	Max	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	2453.0	1741.0	3364.0	2315.0
16/16	Average	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1754.32	1741.0	2554.8533	2315.0
30	Std Dev	0	0	0	0	0	0	82.5749	0	276.6442	0
30	Min	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2315.0	2315.0
	Max	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	3410.0	2315.0
16/16	Average	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2440.7166	2315.0
30	Std Dev	0	0	0	0	0	0	0	0	195.8497	0
40	Min	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2315.0	2315.0
	Max	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	3028.0	2315.0
16/16	Average	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2375.1166	2315.0
30	Std Dev	0	0	0	0	0	0	0	0	122.2043	0
30	Min	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2315.0	2315.0
	Max	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	2144.0	1741.0	3120.0	2315.0
20/20	Average	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1742.3433	1741.0	2388.7366	2315.0
40	Std Dev	0	0	0	0	0	2157.0	23.2672	0	143.089 6	0
40	Min	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2315.0	2315.0
	Max	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	3028.0	2315.0
20/20	Average	1270.0	1270.0	1197.0	1197.0	2157.0	2157.0	1741.0	1741.0	2349.3566	2315.0
40	Std Dev	0	0	0	0	0	0	0	0	85.9779	0

The parameters P_m, P_c, and $it_{\max}$ are varied for GA, while N_{ant} and $it_{\max}$ are varied for ACO

Table 4 Number of tests where the best solutions is found by varying the parameters P_m, P_c and $it_{\max}$ for GA, while N_{ant}, $it_{\max}$, α, and β for ACO (best results in bold)

ACO hyperparameters α = 0.5, β = 1.5 α = 1, β = 1 α= 1.5, β = 0.5		Number of tests where the best solution is found				
N_{ant} $[P_m, P_c]$	$it_{\max}$	4 Odds	6 Odds	8 Odds	10 Odds	12 Odds
		299/300	294/300	281/300	**250/300**	**217/300**
10	10	299/300	298/300	**287/300**	241/300	191/300
		300/300	**299/300**	282/300	214/300	118/300
6/6	10	**300/300**	298/300	229/300	60/300	10/300
		299/300	293/300	**292/300**	277/300	246/300
10	20	**300/300**	296/300	288/300	**279/300**	**250/300**
		300/300	299/300	284/300	260/300	225/300
6/6	20	**300/300**	**300/300**	278/300	144/300	43/300
		298/300	298/300	285/300	**283/300**	**253/300**
20	10	299/300	298/300	288/300	282/300	238/300
		300/300	**300/300**	**291/300**	274/300	204/300
10/10	10	**300/300**	**300/300**	268/300	204/300	23/300
		299/300	298/300	291/300	279/300	269/300
20	20	**300/300**	299/300	293/300	**280/300**	**273/300**
		300/300	**300/300**	294/300	272/300	259/300
10/10	10	**300/300**	**300/300**	**296/300**	202/300	61/300
		297/300	300/300	295/300	290/30	272/300
20	30	**300/300**	299/300	294/300	285/300	**273/300**
		300/300	**300/300**	296/300	**287/300**	253/300
10/10	30	**300/300**	**300/300**	**300/300**	253/300	117/300
		297/300	300/300	291/300	285/30	**279/300**
30	20	299/300	300/300	294/300	**287/300**	276/300
		300/300	300/300	294/300	280/30	271/300
15/15	20	**300/300**	300/300	**299/300**	249/30	96/300
		299/300	295/300	296/300	**288/30**	275/300
30	30	**300/300**	297/300	295/300	286/300	**277/300**
		300/300	**300/300**	296/300	277/300	266/300
15/15	30	**300/300**	**300/300**	**300/300**	278/300	165/300
		299/300	**300/300**	**300/300**	**293/300**	**281/300**
30	40	299/300	299/300	298/300	292/300	277/300
		300/300	**300/300**	298/300	290/300	270/300
16/16	40	**300/300**	**300/300**	**300/300**	288/300	205/300
		300/300	297/300	296/300	288/300	**280/300**
40	30	300/300	298/300	296/300	288/300	272/300
		300/300	**300/300**	299/300	**289/300**	270/300
20/20		300/300	**300/300**	**300/300**	**289/300**	188/300
		300/300	298/300	298/300	290/300	**284/300**
40	40	300/300	**300/300**	299/300	288/300	281/300
		300/300	299/300	299/300	291/300	276/300
20/20		300/300	**300/300**	**300/300**	**292/300**	227/300

6.6 Experimental results

Tables 2, 3, and 4 show the results of experiments 1, 2, and 3, respectively. In Table 2, the number of tests where GA and ACO-CPP found the best solution is represented. In Table 3, the graph augmentations' minimum, maximum, average, and standard deviation. Finally, in Table 4, the number of tests where the best solution was found by GA and ACO-CPP while changing α and β.

Experiment 1: test results demonstrate that ACO-CPP exhibits effectiveness and can converge toward the best solution with success rates of 91% when the parameters N_{ant} and it_{max} are optimized according to the computational complexity of the problem. Furthermore, comparing meta-heuristics, it is noteworthy that GA excels at finding the global optimum when the number of solutions to explore is limited. In contrast, ACO-CPP consistently performs well in finding the global optimum, especially in scenarios involving a wide range of potential solutions. Experiment 1 reveals that ACO-CPP may not achieve the global optimum as effectively as GA when dealing with a relatively small number of possible solutions. However, it demonstrates high effectiveness when the number of solutions increases significantly.

Experiment 2: test results show that ACO-CPP is more efficient and consistent than GA when it comes to obtaining the best solutions in different runs. It consistently demonstrates an equal or significantly lower average best solution for all combinations and an equal or lower standard deviation.

Experiment 3: test results seem to suggest that the proposed experimental setup is more easily solvable with pheromone-based influence when the number of odd nodes is less than 10. Conversely, it appears more easily solvable with heuristic influence or a balanced combination of heuristics and pheromones when the number of odd nodes equals 12. The case with 10 nodes does not seem to exhibit a clear preference.

In our interpretation, the first experiment shows that when the hyperparameter settings are not optimized, ACO-CPP exhibits advantages for complex problems but also weaknesses when dealing with problems involving a limited number of solutions. The second experiment demonstrates that ACO-CPP is a consistent algorithm, a desirable meta-heuristic characteristic. Lastly, the third experiment reveals that ACO-CPP is suitable when hyperparameters are well set, even for problems with limited solutions. Additionally, it unveils some other very interesting characteristics: when an ant is at a point and has to decide which point to move to (a couple of nodes), it has several choices to consider. The ant's decision is influenced by two forces, each contributing to the choice. The first force that can be seen as a "convenience force" is related to the proximity of the nodes (the shortest path), and the closer two nodes are, the higher the likelihood of selecting them (heuristic force). The second force that can be seen as a "past experience force" is associated with the number of ants that have previously passed through a particular couple of nodes and found it to be a good choice; the higher this count, the greater the probability of selecting that node (pheromone force).

In the experiment, assigning a higher value to α instead of β is meant to give more importance to the pheromone force in the first case and more importance to the heuristic force in the second one. The results of the ACO experiments reveal a certain pattern in the quality of solutions. This pattern suggests that, in general, for problems with a limited number of solutions to explore, giving more importance to the pheromone force is advantageous. When the number of odd nodes is 10, there is not one force that outperforms the others. Ultimately, when the number of possible solutions becomes significantly larger, it appears that a heuristic or a balance between heuristic and pheromone forces is more favorable to choose. So, when the ants give more weight to the pheromone force, graphs with fewer than 10 nodes are better solved (relying on past ant experiences). When more weight is given to the heuristic force or a balance between the two is struck, more complex graphs are solved more effectively.

GA finds solutions using three main operators: mutation, crossover, and reproduction, each with its distinct way of exploring the search space. Mutation can be seen as a "random exploration force", capable of exploring different regions of the search space without a specific logic. Reproduction can differently be seen as a "past experience force", favoring solutions that have proven beneficial in the past. Finally, crossover, similarly to reproduction, employs the "past experience force" to conduct local searches and explore the solution space.

A meta-heuristic can be more valuable than others in different contexts, depending on how well its exploration scheme aligns with the structure of the problem's solutions. Therefore, ACO-CPP appears to perform well, as GA, in problems with a limited number of solutions when pheromone and heuristic parameters are properly configured. Above all, ACO seems to be well-suited to solving CPP due to its intrinsic nature for problems with a high number of nodes generating a large number of potential solutions.

7 Conclusions

This paper proposes a solution to the Chinese Postman Problem using an Ant Colony Optimization Algorithm (ACO-CPP). ACO has demonstrated successful applications in both academic and real-world contexts. While it has proven to be a highly effective method for addressing route problems and other NP-hard category problems over time [4], to the best of our knowledge, it has never been tested on the CPP problem.

In this work, our ACO-CPP is evaluated through experiments conducted on connected undirected graphs with randomly generated connections between nodes and edge weights randomly ranging from 1 to 1000. To assess its consistency and effectiveness, ACO-CPP is compared with a Genetic Algorithm (GA) and a recursive algorithm in three separate experiments: (1) recursive-ACO-GA comparisons involving randomly generated graphs to find the global optimum; (2) statistic comparisons between ACO and GA using specifically generated graphs; (3) recursive-ACO(α/β)-GA hyperparameter comparisons over randomly generated graphs for the attainment of the global optimum.

The first and third experiments show the algorithm is efficient. With the correct choice of hyperparameters, it can match the performance of GA when dealing with problems with a limited number of solutions to explore (less than 10 odd nodes). However, it surpasses GA when addressing problems with a significantly larger number of solutions to explore, achieving the global optimum with a high likelihood (greater than 91%). Additionally, the second experiment demonstrates that ACO-CPP exhibits better consistency than GA across different runs.

In the context of the experiments, both GA and ACO appear to perform well when the number of solutions is relatively low, likely because there is no meta-heuristic search method that outperforms significantly. However, as the number of odd nodes increases, GA's performance seems to decline, leading solutions toward local minima. On the contrary, ACO appears more adaptable, possibly due to its pheromone-heuristic approach, making it better suited to tackle the context.

As indicated in Equation (4), the number of solutions to explore increases drastically with the increasing number of pairs of odd nodes in the graph. The real novelty of our ACO-CPP is its ability to address problems with many solutions to explore and to obtain the global optimum by exploring only a reduced portion of the search space, especially in a context like that of CPP. This is achieved while demonstrating superior performance and consistency compared to the other method proposed (GA).

The presented ACO-CPP highlights all the meta-heuristic strengths because, thanks to the pheromone updating strategy, the artificial ants converge toward finding the global optimal solution for a high percentage of attempts while keeping the number of generated solutions low. At the same time, it contains the typical meta-heuristic weaknesses such as the risk of falling into local optima or the difficulty in deciding which are the best parameters to choose [49].

Due to its algorithmic structure, ACO-CPP seems suitable for solving problems such as Directed Chinese Postman Problem (DCPP) and Mixed Chinese Postman Problem (MCPP). Future works will be aimed at designing new experiments to measure its performance in this context. This will involve adapting the recursive algorithm to these contexts and proposing new meta-heuristics to solve such problems for highly complex graphs.

Appendix A: graphs used for experiment 2

See Fig. 6.

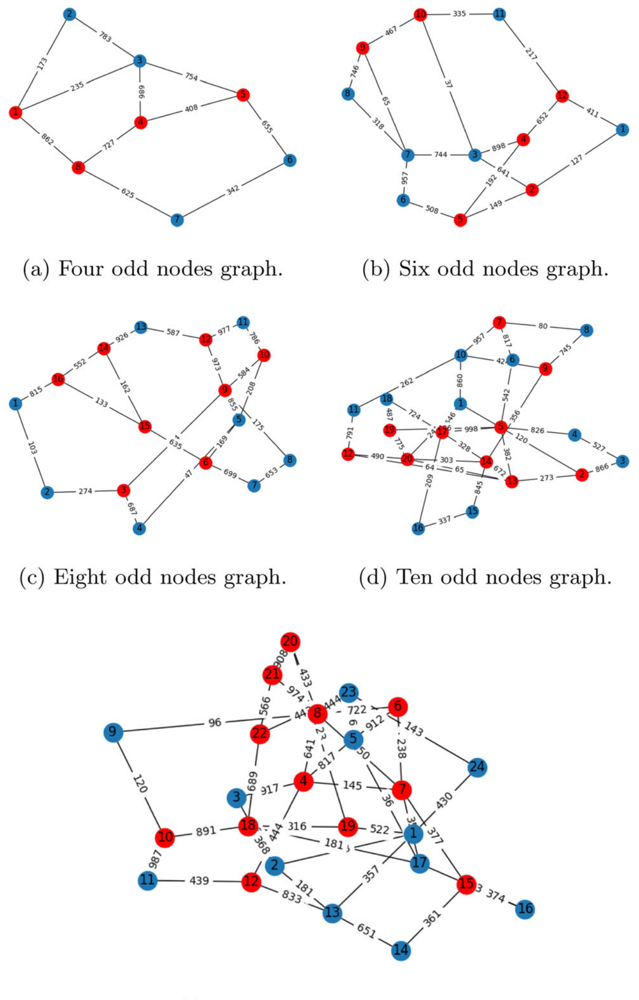

(a) Four odd nodes graph.
(b) Six odd nodes graph.
(c) Eight odd nodes graph.
(d) Ten odd nodes graph.
(e) Twelve odd nodes graph.

Fig. 6 Graphs used for experiment 2

Acknowledgements The authors are deeply thankful to the Grant Office of the University of Foggia for their support. Furthermore, they sincerely thank Dr. Raffaele Sgarro for his precious contribution to reviewing the code scripts.

Funding Open access funding provided by Università di Foggia within the CRUI-CARE Agreement. Research financially supported by: Project MiSE-DGPIIPMI - Artificial Intelligence to support the digitalization and industrial engineering process - 2021.

Data availability All data generated or analyzed during this study are included in this published article.

Declarations

Conflict of interest The authors have no actual or potential conflict of interest in relation to this article.

References

1. Kwan M (1962) Graphic programming using odd or even points. Chinese Math 1:273–277
2. Jiang H, Kang L, Zhang S, Zhu F (2010) Genetic algorithm for mixed chinese postman problem. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol 6382 LNCS, pp 193–199. 10.1007/978-3-642-16493-4_20
3. Filho MG, De Ávila Ribeiro Junqueira R (2010) Chinese postman problem (cpp): solution methods and computational time. Int J Logist Syst Manage 7(3):324–344
4. Dorigo M, Birattari M, Stützle T (2006) Ant colony optimization artificial ants as a computational intelligence technique. IEEE Comput Intell Mag 1(4):28–39
5. Hua J, Li-shan K (2003) Genetic algorithm for chinese postman problems. Wuhan Univ J Nat Sci 8(1):316–318
6. Eiselt HA, Gendreau M, Laporte G (1995) Arc routing problems, part i: the chinese postman problem. Oper Res 43(2):231–242
7. Edmonds J, Johnson EL (1973) Matching, euler tours and the chinese postman. Math Programm 5(1):88–124
8. Larson RC, Odoni AR (1981) Urban operations research vol. monograph
9. Christofides N, Benavent E, Campos V, Corberán A, Mota E (1984) An optimal method for the mixed postman problem. In: Proceedings of the 11th IFIP conference copenhagen system modelling and optimization, pp 641–649. Springer, Denmark, 25–29 July 1983
10. Galil Z, Micali S, Gabow H (1986) An o(ev\logv) algorithm for finding a maximal weighted matching in general graphs. SIAM J Comput 15(1):120–130
11. Derigs U, Metz A (1991) Solving (large scale) matching problems combinatorially. Math Programm 50(1):113–121
12. Lawler EL (2001) Combinatorial optimization: networks and matroids. Courier Corporation, New York
13. Yang J, Huang K, Yin Z, Cui J (2018) The chinese postman problem based on molecular beacon strand displacement. In: 2018 14th International conference on natural computation, fuzzy systems and knowledge discovery (ICNC-FSKD), pp 519–523. IEEE. 10.1109/FSKD.2018.8686916
14. Shen L, Tao H, Ni Y, Wang Y, Stojanovic V (2023) Improved yolov3 model with feature map cropping for multi-scale road object detection. Measure Sci Technol 34(4):045406
15. Stojanovic V, Nedic N (2016) A nature inspired parameter tuning approach to cascade control for hydraulically driven parallel robot platform. J Opt Theory Appl 168:332–347
16. Zhuang Z, Tao H, Chen Y, Stojanovic V, Paszke W (2022) An optimal iterative learning control approach for linear systems with nonuniform trial lengths under input constraints. IEEE Trans Syst Man Cybernet Syst
17. Wu C, Fu X (2020) An agglomerative greedy brain storm optimization algorithm for solving the tsp. IEEE Access 8:201606–201621
18. Emambocus BAS, Jasser MB, Hamzah M, Mustapha A, Amphawan A (2021) An enhanced swap sequence-based particle swarm optimization algorithm to solve tsp. IEEE Access 9:164820–164836
19. Sathya N, Muthukumaravel A (2015) A review of the optimization algorithms on traveling salesman problem. Ind J Sci Technol 8(29):1–4
20. Antunes AR, Matos MA, Rocha AMA, Costa LA, Varela LR (2022) A statistical comparison of metaheuristics for unrelated parallel machine scheduling problems with setup times. Mathematics 10(14):2431
21. Wang W, Zhao J, Huang J (2020) Improved ant colony genetic algorithm for solving traveling salesman problem. J Phys Conf Ser 1693:012085. IOP Publishing
22. Stojanovic V, Nedic N (2016) A nature inspired parameter tuning approach to cascade control for hydraulically driven parallel robot platform. J Opt Theory Appl 168:332–347
23. Xia D, Quan W, Wu T (2022) Optimizing functional near-infrared spectroscopy (fnirs) channels for schizophrenic identification during a verbal fluency task using metaheuristic algorithms. Front Psychiatry 13:939411
24. Tan KC, Lee LH, Zhu Q, Ou K (2001) Heuristic methods for vehicle routing problem with time windows. Artif Intell Eng 15(3):281–295
25. Abdor-Sierra JA, Merchán-Cruz EA, Rodríguez-Cañizo RG (2022) A comparative analysis of metaheuristic algorithms for solving the inverse kinematics of robot manipulators. Res Eng 16:100597
26. Gagnon I, April A, Abran A (2020) A critical analysis of the bat algorithm. Eng Rep 2(8):12212
27. Nejad AS, Fazekas G (2022) Solving a traveling salesman problem using meta-heuristics. IAES Int J Artif Intell (IJ-AI) 11(1):41
28. Sudholt D, Thyssen C (2012) Running time analysis of ant colony optimization for shortest path problems. J Discrete Algorithms 10(1):165–180
29. Di Caprio D, Ebrahimnejad A, Alrezaamiri H, Santos-Arteaga FJ (2022) A novel ant colony algorithm for solving shortest path problems with fuzzy arc weights. Alexandria Eng J 61(5):3403–3415
30. Christofides N (1976) Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group

31. Gunderson DS (2014) Handbook of mathematical induction: theory and applications. CRC Press, New York, p 240. 9781420093650
32. Hein JL (2015) Discrete structures, logic, and computability, example 3: the handshaking problem, p 703. Jones & Bartlett Publishers, LLC. ISBN: 9781284070408
33. Simenthy LJ, Bobanand R, Soumya Krishnan M (2015) A comparison based analysis of euler circuit finding algorithms. Int J Appl Eng Res 10(55):2511–2514
34. Dorigo M, Di Caro G, Gambardella LM (1999) Ant algorithms for discrete optimization. Artif Life 5(2):137–172
35. Liang Y-C, Minanda V, Gunawan A (2022) Waste collection routing problem: a mini-review of recent heuristic approaches and applications. Waste Manage Res 40(5):519–537
36. Wang C, Zhu R, Jiang Y, Liu, W., Jeon, S.-W., Sun, L., Hang, H (2023) A scheme library-based ant colony optimization with 2-opt local search for dynamic traveling salesman problem. CMES Comput Model Eng Sci 135(2)
37. Ky Phuc PN, Phuong Thao NL (2021) Ant colony optimization for multiple pickup and multiple delivery vehicle routing problem with time window and heterogeneous fleets. Logistics 5(2):28
38. Azeez MI, Abdelhaleem A, Elnaggar S, Moustafa KA, Atia KR (2023) Optimized sliding mode controller for trajectory tracking of flexible joints three-link manipulator with noise in input and output. Sci Rep 13(1):12518
39. Mehmood N, Umer M, Asgher U (2023) Application of hybrid sfla-aco algorithm and cam softwares for optimization of drilling tool path problems. SN Appl Sci 5(2):61
40. Al Bataineh A, Kaur D, Jalali SMJ (2022) Multi-layer perceptron training optimization using nature inspired computing. IEEE Access 10:36963–36977
41. Singh H, Tyagi S, Kumar P, Gill SS, Buyya R (2021) Metaheuristics for scheduling of heterogeneous tasks in cloud computing environments: analysis, performance evaluation, and future directions. Simul Modell Pract Theory 111:102353
42. Beschi M, Mutti S, Nicola G, Faroni M, Magnoni P, Villagrossi E, Pedrocchi N (2019) Optimal robot motion planning of redundant robots in machining and additive manufacturing applications. Electronics 8(12):1437
43. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybernet Part b (cybernetics) 26(1):29–41
44. Stützle T, Hoos HH (2000) Max-min ant system. Future Generation Comput Syst 16(8):889–914
45. Dorigo M, Gambardella LM (1997) Ant colonies for the travelling salesman problem. Biosystems 43(2):73–81
46. Gambardella LM, Dorigo M (1996) Solving symmetric and asymmetric tsps by ant colonies. In: Proceedings of IEEE international conference on evolutionary computation, pp 622–627. IEEE
47. Wahde M (2008) Biologically inspired optimization methods: an introduction. WIT press, Boston
48. Biggs N, Lloyd EK, Wilson RJ (1986) Graph theory, 1736–1936. Oxford University Press, Oxford
49. Jones DF, Mirrazavi SK, Tamiz M (2002) Multi-objective metaheuristics: an overview of the current state-of-the-art. Eur J Oper Res 137(1):1–9