IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 19, 2020, accepted December 28, 2020, date of publication January 1, 2021, date of current version January 11, 2021.

Digital Object Identifier 10.1109/ACCESS.2020.3048774

On Randomness and Structure in Euclidean TSP Instances: A Study With Heuristic Methods

GLORIA CERASELA CRIŞAN[1], ELENA NECHITA[1], AND DANA SIMIAN[2]
[1]Department of Mathematics and Informatics, Vasile Alecsandri University of Bacău, 600115 Bacău, Romania
[2]Research Center in Informatics and Information Technology, Lucian Blaga University of Sibiu, 550024 Sibiu, Romania

Corresponding author: Gloria Cerasela Crişan (ceraselacrisan@ub.ro)

The work of Gloria Cerasela Crişan and Elena Nechita was supported by the Ministry of Education and Research through the National Council for the Financing of Higher Education, Romania, under Grant CNFIS-FDI-2020-0461. The work of Dana Simian was supported by the Lucian Blaga University of Sibiu and the Hasso Plattner Foundation under Grant LBUS-RRC-2020-01.

ABSTRACT Prediction of the quality of the result provided by a specific solving method is an important factor when choosing how to solve a given problem. The more accurate the prediction, the more appropriate the decision on what to choose when several solving applications are available. In this article, we study the impact of the structure of a Traveling Salesman Problem instance on the quality of the solution when using two representative heuristics: the population-based Ant Colony Optimization (ACO) and the local search Lin-Kernighan (LK) algorithm. The quality of the result for a solving method is measured by the computation accuracy, which is expressed using the percent error between its solution and the optimum one. We classify the instances in structured, semi-structured, and unstructured and perform a between classes and inside-classes analysis. All the structured instances were solved to optimality by the ACO implementation, which was not the case for the LK application. On small random instances, the ACO implementation used in this paper also optimally found the solutions. We show that the quality of the results on semi-structured and unstructured instances can be predicted using some instance parameters when using the ACO implementation. Using the same parameters, the accuracy of the solutions provided by the Lin-Kernighan application cannot be predicted. We also propose several new structured, clustered, and unstructured 2D Euclidean Traveling Salesman Problem instances that can be used by the research community for further investigations.

INDEX TERMS Ant colony optimization, traveling salesman problem, Euclidean norm, Lin-Kernighan method.

I. INTRODUCTION

This work focuses on the study of the influence of structural features of Traveling Salesman Problem (TSP) instances on computational performances measured by solution quality when solving TSP using two representative heuristics: Ant Colony Optimization (ACO) [1] and Lin-Kernighan method (LK) [2]. By solution quality, we understand the solution accuracy given by the percent error between the solution and the optimum one.

When choosing a specific solving method for a given problem, we have to consider several aspects. When no information on the problem instance is known, large available computational resources may orient the user's decision toward expensive, intensive solvers that provide an optimum solution. For example, for an NP-hard problem as is the Traveling Salesman Problem (TSP), an exact, very efficient application used 84.8 CPU equivalent-years on a single Intel Xeon 2.8 GHz processor for the Sweden instance [3]. In the opposite case, when resources are scarce, a ''light'' solver may be accepted, providing an acceptable solution. Approximation or heuristic methods, which offer ''good-enough'' solutions, are recommended in this case. Sometimes a small threshold for accepting non-optimal solutions could tremendously ease the computational burden [4].

To solve the TSP means to find one least-weighted cycle connecting all the vertices of a complete graph with weights on edges [5]. Euclidean TSP considers that each edge weight represents the Euclidean distance between the corresponding incident vertices. Certain solver could work better on a specific sub-class of the TSP problem, exploiting the problem specificity. Therefore, choosing a solver should be the

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

result of a more informed decision. For example, for the TSP instances where weights satisfy the triangle inequality, the problem is approximated within a factor of 3/2. TSP with Euclidean distances admits a polynomial-time approximation scheme [6].

Previous works studied the impact of the problem characteristics on the computational cost spent for solving it. Besides the *size*, which is expected to determine in most cases the exponential growth of the solving time, some *structural* features could also impact the difficulty of exact methods. For example, [7] investigated the impact of the coefficient of variation of the (integer, randomly generated) distance matrix on an exact solving method for TSP. In [8] arbitrarily large Euclidean TSP instances that are always solved to optimality by the Nearest-Neighbor heuristic were described. An index for classifying small 2D Euclidean TSP instances as clustered, random or regular was proposed in [9]. This index was used for studying the human performances for solving small TSP instances [9], [10].

In the existing literature, there were identified four main directions in which the influence of various features on instances of different optimization problems could be important [11]: algorithm selection, parameter tuning, gaining information on the instance difficulty and algorithm computational performances, and generation of new instances that could be used as benchmarks. In the last years, different Machine Learning (ML) approaches were used to study algorithm selection for the TSP [12] and to predict the run-time of a TSP algorithm for solving a given instance. Run-time is one of the stopping criteria used for approximate sub-optimal algorithms. The desired precision for a solution could not be reached because of the limited run-time imposed [11], [13]. Neural networks proved good results in making predictions of performance and in classifying the instances into difficult and easy to solve [14], but linear regression, ridge regression, regression trees, Gauss regression, random forests were also used [11].

The algorithm LK and its variant LKH-2 [15], the multi-agent based solver MAOS [16], the evolutionary algorithm EAX using edge assembly crossover [11], [12], [14] and evolutionary algorithms combining edge assembly crossover and partition crossover [17] are on the top of approaches used for testing the TSP algorithm selectors designed during the time.

The per-instance algorithm selection approach was adapted to the TSP problem for the first time in [18]. Creating specific instances for a study is not new. In [19] a set of artificially created instances, with different levels of difficulty were used to study the influence of instance features on the performances of two variants of the LK algorithm. For algorithm selection problems a large set of features characterizing both the instances and the algorithm steps allows a better decision. However, when focusing on obtaining new insights into the influence of different features on specific algorithm performance, using only a few features could be also relevant [13].

Our study aims to gain insights into TSP instances difficulty based on structural features. More precisely we want to see to what extent the structural properties influence the hardness of TSP instances. The structural properties depend on the instances generation process and are evaluated using the coefficient of variation [7] and the regularity index [9].

In our investigation, we use a set of 2D Euclidean TSP instances classified by us into three classes: structured, semi-structured, and unstructured.

The set of structured instances is composed of 50 tetrahedron instances, *Tnm,* from [20], 18 instances generated based on the *NPeano* curve from [21], and 4 instances generated based on regular tessellation with hexagons. The tetrahedron instances *Tnm* proved to be difficult when solved with Concorde, currently considered the best exact solver for symmetric TSP. One generic *Tnm* instance has n vertices on each side of an equilateral triangle and m vertices on its medians and is described by the total number of vertices. For example, *Tnm100* has 100 vertices. The *NPeano* curve was generated in [21] based on the well-known space-filling Peano framework. It allows a semi-regular plane tessellation with octahedrons and squares. The semi-structured instances are 19 national instances with vertices being real localities [3] and 20 clustered instances randomly generated in a $10^6 \times 10^6$ square. Unstructured instances are 10 uniformly generated instances in a square with side 1000, and 20 ones of the same type generated in a $10^6 \times 10^6$ square.

We report on the computational performance of two implementations (one for ACO and the other one for the LK heuristics) when considering structured, semi-structured, and unstructured instances as well as on different sub-classes of the structured instances used in this article. The computational performance is assessed using the *gap* parameter [12]. The results show that for the 2D Euclidean TSP, the intrinsic structure of the instance influences the quality of the results. The ACO version used in this paper solved to optimality all the structured instances, while the LK implementation optimally solved only 43% of them. On the other instances we show that the ACO accuracy can be predicted with enough confidence using three parameters (details are given in Section II.D), but these parameters do not influence the LK accuracy. This proves that these three parameters are not enough for making good predictions for both methods.

This paper is structured as follows: in Section II we present the fundamentals on which the experiments performed in this article are based. The formulation of the Traveling Salesman Problem, the presentation of ACO and LK heuristics, and the instances used in our experiments can be found in this section. Section III describes and interprets the results of the experiments performed. Conclusions are given in Section IV.

II. MATERIALS AND METHODS

A. THE TRAVELING SALESMAN PROBLEM (TSP)

Dantzig gives in [22] one of the simplest Traveling Salesman Problem (TSP) specifications: ''In what order should

a traveling salesman visit n cities to minimize the total distance covered in a complete circuit?''. When described using the Graph Theory notions, TSP supposes that a complete weighted graph is given and seeks for a least weight Hamiltonian cycle [5].

In terms of linear programming, one of the most known TSP specifications is given in [23]. It considers n points with known distances between any two of them. Let d_{ij} be the distance between the points i and j, with $1 \leq i, j \leq n$, and the n^2 binary variables x_{ij}, defined as 1 if the path goes from i to j and 0 otherwise. Then the TSP is given by the following integer linear programming problem [23]:

$$min \sum_{i,j=1}^{n} d_{ij} x_{ij} \tag{1}$$

under the following constraints:

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \forall 1 \leq j \leq n \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall 1 \leq i \leq n \tag{3}$$

$$\sum_{i,j \epsilon S} x_{ij} \leq |S| - 1, \quad \forall S \subset \{1, 2, \ldots, n\}, \; 2 \leq |S| \leq n-1 \tag{4}$$

The objective function from (1) is the total length of a generic set of arcs. The constraints (2) / (3) force that for any point, the considered set of arcs contains exactly one arc which enters into/exits from that point. The constraints (4) forbid early closings, meaning that multiple, not connected circuits are not permitted.

TSP is an old and widely approached optimization problem, with many variants, multiple proposed solutions for each variant, and broad applications. Different criteria are used to classify TSP variants.

The properties of the matrix $(d_{ij})_{1 \leq i,j \leq n}$ provide a first classification criterion. *Symmetric* TSP considers that the distance matrix d is symmetric, i.e. for each pair of vertices, the distance in both ways is the same [24]. *Asymmetric* TSP supposes different distances for at least one pair of vertices and models traffic situations as one-way roads, bottlenecks, etc. [25]. The *metric* TSP is a symmetric TSP with distances obeying the triangle inequality [26]. If the distances are computed using a Euclidean metric, then the metric TSP is *Euclidean* [27]. The most common Euclidean TSP uses the 2D norm, due to its connections with real-life situations when routes have to be found on a map.

All the instances approached in this article are 2D Euclidean *with integer distances, i.e.* the distances are integer approximations of 2D Euclidean distances. The *size* of a TSP instance is given by n.

Specific constraints were designed to adapt the TSP formulation for solving practical problems from different fields. The most natural TSP applications are in Transportation and Logistics when efficient routes need to be found for connecting cities or to deliver products [28]. Formulations based on profit maximization or cost minimization were introduced, leading to variations of classical TSP. For example, *Profitable Tour Problem* aims to minimize the difference between travel costs and profit; *Travelling Purchaser Problem* aims to minimize the routing and the purchasing costs when selecting a subset of markets [29]. Additional constraints on the time window in which every node can be visited are introduced in the *TSP with Time Windows* variant [30]. Drone integration in TSP is modeled as *TSP with Drone* (TSP-D) [30]. In this case, the traveling salesman and the drone form a cooperating system that starts and arrives from/at a specific point called *depot*. Each *client* is visited by either the traveling salesman or by the drone, which could perform independently and simultaneously. The distances between the *points* (either clients or depot) reflect the travel time between them, which are different for the human and for the drone. The goal of TSP-D is to visit all the clients in the shortest possible time. Other applications with specific adaptations of the TSP can be found in industrial manufacturing [31], genetics [32], telecommunications [33], etc.

Uncertainty may be considered when defining TSP variants. It is integrated into TSP specification for modeling complex real-life situations. *Probabilistic TSP* (PTSP) [34] is obtained when the points are clients which do not need everyday visits and the traveling salesman is located in a special location (the depot). In PTSP, each client i has to be visited with probability p_i and every day each client requests (or not) a visit from the traveling salesman. The depot is always the starting and also the ending point of the tour performed each day. The goal of PTSP is to find a complete cycle with the least expected length. If the distance matrix contains triangular fuzzy numbers and the length of a cycle is computed by using the classical fuzzy operator for addition $\oplus$, then we obtain the *Fuzzy TSP* [35]. An *Uncertain Multi-objective Traveling Salesman Problem* is introduced in [36].

As TSP is an NP-hard problem [37], the exact methods are expected to become intractable when the problem size (the number of vertices) increases. The best exact solver for *symmetric* TSP is currently considered to be Concorde [38], which is publicly available at [39]. We used it for computing the optimum solutions on the new instances *Peano*, *Hex,* and *Rand* that we generated to carry out this study (more details are given in Section III.A).

Heuristic methods are widely applied for quickly obtaining near-optimum solutions. *Construction* heuristics based on Greedy strategies are usually combined with local search methods as *2-opt* or *3-opt*, to obtain solutions with increased precision. The Nearest-Neighbor method is one of the simpler greedy heuristic algorithms for TSP [40]. It builds a path in a ''step by step'' mode, starting from an arbitrary vertex, adding the nearest available one, and finally returning from the starting vertex. Other greedy heuristic method starts by adding the shortest edge to the solution and continues to add edges in increasing order of their lengths until a solution is constructed [40]. Each edge is added if it does not prematurely close the circuit or it does not produce a vertex with degree 3. Therefore, this Greedy method constructs the solution by adding separate edges that get connected while the algorithm

evolves. The local search *2-opt* starts from a solution and tries to improve its quality by deleting a pair of edges and reconnecting correctly the remaining parts. Similarly, *3-opt* deletes 3 edges and seeks for a better solution containing what was left.

An adaptable local search heuristic with very good results in solving symmetric TSP is the Lin-Kernighan method [2]. It starts from a feasible solution and determines the value for a parameter k which bounds the iterative improvements using *2-opt*, *3-opt*,..., *k-opt*. More heuristics for TSP can be found in [41]–[43].

Metaheuristic methods are frameworks that combine many heuristics for solving a specific class of problems. Different *classical* metaheuristic algorithms (Tabu Search, Simulated Annealing), *evolutionary* algorithms (Genetic Algorithms, Scatter Search) and *nature-inspired* algorithms (Ant Colony Optimization, Cuckoo-Search, Firefly, Snail, Monarch Butterfly Optimization, Harmony Search, Earthworm Optimization, Elephant Herding Optimization, Moth Search algorithms, African Buffalo Optimization, etc.) have been used to solve problems derived from the TSP. These algorithms hold mechanisms for escaping from the local optima when they are searching for global optima. For a comprehensive presentation and comparison of the most used metaheuristic algorithms for various types of TSPs, we refer the reader to [44]–[49] and the references therein.

In this article, we used ACO and LK as reference heuristics for solving different TSP instances and compared the computational accuracy. These two heuristics are representative of the two main categories of heuristics [41]: problem–independent heuristics category (ACO) and local search heuristics exploiting specific information on the problem domain (LK). In many studies, ACO is considered for comparison when evaluating different new proposed metaheuristics (e.g. Discrete Monarch Optimization [50], African Buffalo Optimization [51], Earthworm optimization [47].) ACO and LK are also representatives because many other metaheuristics are hybridized using ACO or LK in current state-of-the-art approaches. For example, the Multi-Neighborhood Search (MNS) inspired by [52] was used in [53] to hybridize an implementation of Population-based ACO (PACO). This comprehensive work investigated 193 setups for hybrid population-based and local search algorithms, proposing *TSPSuite*, a framework for analyzing multiple attributes of multiple algorithms on multiple instances. From all the 193 hybrid algorithms analyzed in [53], 6 different methods using PACO were ranked in the first 6 places. This is another reason why we considered the ACO heuristic as a basis in our study. A successful hybrid approach using Lin-Kernighan and Hill Climbing outperforms the state-of-the-art methods for the *Traveling Thief Problem*, which combines TSP with *0-1 Knapsack Problem* [54].

B. ANT COLONY OPTIMIZATION

Ant Colony Optimization (ACO) [1], [55] is a metaheuristic that mimics the foraging behavior of real ant colonies. It captures the pheromone communication system between ants. In the beginning, ants take random ways when searching for food. They mark the path between the nest and food source by laying pheromone on the ground. The other ants will follow the path marked with the greatest quantity of pheromones. The shortest path will accumulate in time the largest quantity of pheromones being followed by the rest of the ants in the colony.

Three processes are fundamental in an ACO algorithm: *Solutions generation*, *Daemon actions* and, *Pheromone trail update* [55]. Each artificial ant constructs (in parallel with the others) a solution during the *Solution generation* phase. *Daemon Actions* represent the centralized or distributed activities triggered by the current solutions. For example, a local search procedure like 3-opt is applied to the solutions. During the *Pheromone update* phase, the numerical values representing the pheromone quantities are updated. Each process is implemented in a specific way, so devising different ACO algorithms. The general description of ACO [1] is presented in the following.

```
procedure ACO_MetaHeuristic
    while not_termination do
        generateSolutions()
        daemonActions()
        pheromoneUpdate()
    end while
end procedure
```

This very general ACO framework was the starting point for several Ant Algorithms, each one with a specific strategy in constructing the ant paths and also in updating and evaporating the pheromone. When solving TSP using the ACO approach, a fixed number of artificial agents are randomly placed in the vertices of the TSP and each one constructs a cycle traversing all the vertices. The construction is probabilistically guided by the weights of the available edges and also by the quantity of artificial pheromone deposited by ants that previously used those edges. This process is iterated until a specific ending condition is met and the algorithm exits with the best cycle found (the one with minimum total weight) [1].

One of the first ACO algorithms tested on TSP was Ant System (AS) [1], [56]. The probabilistic rule that governs the path construction is given in formula (5). If the ant k is in the vertex i, then the probability that the next vertex in the path is j is given by p_{ij}^k.

$$p_{ij}^k = \frac{{(\tau_{ij})^\alpha}/{(d_{ij})^\beta}}{\sum_{l \in N_i^k} {((\tau_{il})^\alpha}/{(d_{il})^\beta})}, \quad j \in N_i^k \tag{5}$$

In formula (5), the set of the available moves from i is N_i^k. The distance between nodes i and j is d_{ij} and the amount of the pheromones on the edge between vertices i and j is τ_{ij}. The parameters α and β set the importance of the exploitation of previous information (stored in the pheromone matrix τ) vs. the exploration of the graph (using the distance matrix d).

After all ants complete their solutions, the pheromone update is done using the formulae (6) and (7):

$$\tau_{ij} = (1-\rho)\,\tau_{ij} + \rho \sum_k \Delta\tau_{ij}^k \quad \forall i,j \tag{6}$$

$$\Delta\tau_{ij}^k = \begin{cases} 1/L_k & \text{if the ant } k \text{ used the edge } (ij) \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Formula (6) describes the pheromone update rule. After evaporation governed by the parameter ρ, only the used edges receive a pheromone reinforcement. Formula (7) computes the reinforcement corresponding to the specific ant k. It is proportional to the inverse of the length of its path.

Multiple other versions of Ant Algorithms were later described [57], [58], optimizing different aspects of ACO, or created a better ACO type algorithm for a given problem. Improving the behavior and performance of the ACO algorithms has been achieved in various ways and is focused on all phases of the ACO algorithm individually or by the ensemble. Related to the first phase of the algorithm, different restarting procedures when a condition is fulfilled were used for accelerating the convergence and decreasing the computational time [59]. Usual Daemon Actions are based on the knowledge provided by the problem specificity, but can not be exploited by the ants for improving further exploitation process of solutions. If a local search procedure is used in the Damon Actions phase, for improving the constructed solutions, the actions undertaken are hidden from the ants. A Non-DaemonActions procedure was proposed in [60], giving the ants the possibility of learning the modifications applied by the local search procedure to their solutions. More precisely, Non-DaemonActions are swapping many edges with their pheromone concentrations and heuristic information. After a complete tour of the ants, a global updating rule is applied and the Non-DaemonActions procedure is used for improving the best solution of the current iteration. Thus, the exploitation around the iteration-best solution is enhanced.

The pheromone update phase plays a very important role in the algorithm performances. In [58] the pheromone concentration is initialized according to the path information and a dynamic strategy is adopted both for pheromone updating and pheromone evaporation control. Three strategies are proposed in [61] for reducing the current drawbacks of ACO while handling large instances. The first strategy consists in improving the local search performances by using pheromone information. In this way, candidates-set of best possible nodes can be dynamically considered in local search. A new representation for pheromone values, providing linear pheromone space complexity, is proposed as a second strategy. The third strategy focuses on speeding up the ACO algorithm and is based on a new approach for selecting the next node.

MAX-MIN Ant System (MMAS) comes with three modifications when compared to AS:

- The pheromone is forced to belong to a specific interval $[\tau_{min}, \tau_{max}]$. The reason is to maintain a balanced influence of all edges and therefore to induce a balanced exploitation;
- τ_{max} is the initial value of the pheromone on all edges. The reason is to increase the exploration in the initial phase.
- Only the best ant is allowed to deposit pheromone. The reason is to orient the search towards the most promising solutions.

For our experiments we have chosen the ACOTSPQAP software package from [62]. It is an open collection of efficient methods for TSP and Quadratic Assignment Problem (QAP), provided by the ACO designers, with multiple implementations, offering adaptable choices for parameters. Being a population-based method, the execution time for the suggested values of parameters is expected to be larger than the average local search execution time. Among its available algorithms, we used MAX-MIN Ant System (MMAS) hybridized with *3-opt* local search. MMAS is described in many papers as providing very good results [45]. We executed the code with the default values for parameters: $\alpha = 1, \beta = 2$, and $\rho = 0.02$.

C. LIN-KERNIGHAN (LK) METHOD

LK is one of the most effective approaches for solving symmetric TSP [2]. It is a heuristic method that iteratively improves an admissible solution using the interchange strategy from k-opt, but with an adaptive k for each iteration. In each iteration of the improvement process the new solution is computed step by step, successively applying *2-opt*, *3-opt*,..., *k-opt* until no improvement can be done to the initial admissible solution from this iteration.

The LK method starts from an initial solution and makes a first-improvement move when the sequence of opt operations are applied. After a move, the sequence of tests begins again from 2-opt. Iterated LK (ILK) maintains a pool of solutions that are tested for improvements using the opt moves [63]. Chained LK (CLK) applies LK and if no improvement is possible, it perturbs the current solution by using a special 4-opt move, called "double bridge" [64]. Efficient k-opt implementations are proposed in [15].

Currently, LK is considered to be one of the most efficient local search methods. As it works on a single solution, its computational cost is expected to be very low. In this paper we used the online CLK implementation from [39], executed on servers from the Arizona State University. This remote application is efficient, and deterministic if ran with fix seed for the random number generator. It has a downloadable version, with fewer options, but the execution speed on an average PC is the same as the remote execution speed. As the architecture of the remote servers is unknown, no discussion will consider the execution time. We focus instead on the gap, which is sometimes referred to as *accuracy*. But the ACO and LK implementations are not compared one against the other. This study discusses their individual characteristics when solving the instances we approached.

D. TSP INSTANCES CHARACTERIZATION

TSP is proven to be NP-hard [37] and its Euclidean variant is NP-complete [65]. Therefore, the computational effort for exactly solving the problem increases exponentially with the size of the instances. In the last decades, approximate approaches, partially enumerated in Section II.A, gained increased interest in solving TSP. Attempts have been made to test the performances of some of these methods, and to tune their parameters, on different TSP instances [66]–[68]. Runtime and accuracy on a given set of instances have been considered as the main measures of algorithm performance. There are multiple TSP data collections available for testing. Some of the most known such collections are TSPLIB [70] and TSP Test data [71]. In the last decades, the attention was directed to the generation of instances with different hardness levels for testing different approaches for TSP. In [68], using an evolutionary algorithm, instances with different levels of difficulty were generated for testing and parameters tuning of ACO algorithms for the TSP. The instances were classified based on the approximation ratio, i.e. the quotient between the tour length produced by the algorithm and the length of the exact solution generated using the Concorde solver. The higher the approximation ratio, the heavier the instance. The number of nodes, the edge cost distribution, and features related to possible clusters, statistical based uniformity of an instance are also taken into account for hardness characterization of instances [66], [67]. An evolutionary method to evolve TSP instances based on sophisticated mutation operators was proposed in [69]. The method evolved instances with different topologies, and a different behavior against two given solvers (Edge Assembly Crossover - EAX and Modified Lin-Kernighan LKH-2), i.e. instances that are difficult for one solver and easy for the other solver. These kinds of sets of TSP instances allow emphasizing the advantages and disadvantages of different TSP solvers.

In this article, we conduct an empirical study on the influence of instances structure on the quality of the solution of the Euclidian TSP using ACO and LK approaches. To reach our goal we used the instances from [20], [70] and our own generated instances [72], as shown in Section III.

We classify the instances into three categories: structured, semi-structured, and unstructured. In our study, by *structure,* we understand the degree of regularity of the vertex distribution. In the structured instances, the coordinates of the vertices are found using mathematical formulae. The semi-structured instances model the natural distribution of cities in geographic maps. The vertices in unstructured instances are randomly spread in a rectangle.

In our experiments, we used three sets of instances, based on their structure.

a) The *structured* instances are all analytically constructed. Their vertices are deployed using mathematical formulae. Two subsets of structured instances were used. The first subset includes the *Tnm* instances from [20]. The second subset contains new *Peano* instances extracted from a semi-regular tessellation with octahedrons and squares, and also new *Hex* instances, derived from a regular tessellation with hexagons. The new instances, designed specifically for this work, are available on the webpage [72] and represent a contribution to the open datasets available for future investigations. Contributing with new TSP instances to the open datasets is a long-standing preoccupation of the authors [43], [73], [74].

b) The *semi-structured* instances model the positions of some real localities. These instances are considered semi-structured in the literature, as they manifest specific clusterization of natural systems. The set of semi-structured instances is composed of real instances from [20] and clustered instances automatically generated using the *portcgen* application from the 8th DIMACS Implementation Challenge [75].

c) The *unstructured* instances are instances with random points with integer coordinates in squares with different areas. We use the model of Random Uniform Euclidean (RUE) instances [11], [12] generated using *portcgen* [75].

More details on the instances chosen in our experiments are provided in Section III.

In [7], the authors empirically studied the impact of the standard deviation of the distance matrix having the average value 10 on the execution time of an exact method for solving TSP. The sizes of the instances were 16, 32, 48, adapted to computing resources of that time. The results showed the unimodal distribution of the execution time when plotted against the standard deviation of the distance matrix.

The influence of the instance structure on the difficulty to solve TSP by humans was studied in many articles [9], [10]. The measures used for this goal are of interest also for the characterization of the TSP difficulty for computer methods and solvers. Based on their regularity, the TSP instances were classified as highly clustered, random, and highly regular [9], [10]. The regularity of the TSP structure is characterized in [9] by the index R, given in formula (8):

$$R = \frac{2(\sum_{i=1}^{n} r_i)}{\sqrt{A \cdot n}} \tag{8}$$

where r_i is the shortest distance from i to all other vertices, and A is the minimum area occupied by the vertices.

According to [9] the values of the R index for a different type of TSP instances are as follows:

- for highly clustered instances, R is less than 0.24
- for random instances R is very close to 1
- for highly regular instances with vertices uniformly spread, R is greater than 1.76.

We want to emphasize that the classification of instances using the regularity property in the sense given in [9] and based on the R parameter does not overlap our proposed classification of instances in structured, semi-structured, and unstructured, as we can see in Section III.

Another measure to predict the difficulty of a TSP instance is the *normalized knot* (NK) index [76], based on the minimum spanning tree (MST) built for an instance. NK index is the division between the number of the vertices with a degree at least 3 from the MST and the instance size. The authors

TABLE 1. Structured instances *Tnm*.

Name	Size	Optimum	CV	R
Tnm52	52	551609	0.505	1.240
Tnm55	55	605778	0.504	1.218
Tnm58	58	660687	0.503	1.194
Tnm61	61	716131	0.502	1.173
Tnm64	64	770162	0.502	1.153
Tnm67	67	825328	0.501	1.135
Tnm70	70	881036	0.501	1.119
Tnm73	73	893843	0.502	1.093
Tnm76	76	949961	0.501	1.077
Tnm79	79	1006535	0.501	1.062
Tnm82	82	1062686	0.501	1.048
Tnm85	85	1117381	0.500	1.034
Tnm88	88	1172734	0.500	1.021
Tnm91	91	1228726	0.500	1.009
Tnm94	94	1285416	0.500	0.997
Tnm97	97	1342086	0.500	0.980
Tnm100	100	1398070	0.500	0.965
Tnm103	103	1412229	0.500	0.955
Tnm106	106	1469617	0.499	0.944
Tnm109	109	1527709	0.499	0.930
Tnm112	112	1585157	0.499	0.917
Tnm115	115	1641752	0.499	0.905
Tnm118	118	1698486	0.499	0.895
Tnm121	121	1755689	0.499	0.881
Tnm124	124	1813276	0.499	0.870
Tnm127	127	1871162	0.499	0.860
Tnm130	130	1928734	0.499	0.850
Tnm133	133	1944317	0.499	0.845
Tnm136	136	2002445	0.499	0.835
Tnm139	139	2060637	0.499	0.826
Tnm142	142	2118758	0.499	0.817
Tnm145	145	2177169	0.499	0.809
Tnm148	148	2235669	0.499	0.800
Tnm151	151	2293265	0.499	0.792
Tnm154	154	2350345	0.499	0.785
Tnm157	157	2407153	0.499	0.777
Tnm160	160	2463857	0.499	0.770
Tnm163	163	2485463	0.499	0.766
Tnm166	166	2543466	0.499	0.759
Tnm169	169	2600546	0.499	0.752
Tnm172	172	2657369	0.499	0.745
Tnm175	175	2714530	0.498	0.739
Tnm178	178	2771953	0.498	0.733
Tnm181	181	2829701	0.498	0.727
Tnm184	184	2887675	0.498	0.721
Tnm187	187	2945939	0.498	0.715
Tnm190	190	3004259	0.498	0.710
Tnm193	193	3023551	0.498	0.706
Tnm196	196	3081566	0.498	0.701
Tnm199	199	3139778	0.498	0.696

showed that humans find better solutions to 2D Euclidean TSP when the NK index is small.

A more general approach when studying the performance of *local search* heuristics applied to optimization problems is given by the impact of the fitness landscape [77]. For a given problem, the *fitness landscape* a.k.a. *search landscape* is the triplet (*solution space*, *neighborhood operator*, *objective function*). This notion is inspired by the *Adaptive Landscape* diagram from [78]. Current research highlights that the overall structure of the fitness landscape has a clear impact on the performance of local search methods [79]. For example, for a given local search method, the neutral instances were empirically proven to be more difficult. An instance is *neutral* if two neighbor solutions could have the same value of the objective function (i.e. fitness or cost value).

In our work we adapted and developed the methodology from [7] and [9], using three parameters: *size*, the R index, and the coefficient of variation (CV). We note that the *order parameter* used in [7] is identical to the coefficient of variation. We do not investigate the impact of the NK index, because our study considers unstructured, semi-structured, and structured instances, and for structured instances, there are multiple non-isomorphic MST solutions.

We investigated the evolution of the fitness value for 12 instances, four from each group. We adapted the fitness landscape basic idea to ACO, although it is not a local search procedure, but a construction one. Details of this adaptation are given in Section III.C.

III. RESULTS

A. EXPERIMENTAL SETTINGS AND DESIGN

The experiments were conducted under a Linux distribution for a quad-core Intel i3 processor at 2 GHz with 4 GB RAM. The Ant Colony Optimization (ACO) application was run locally with implicit parameters, with 25 artificial ants. The Lin-Kernighan implementation from [39] was remotely used. This application is deterministic, extremely efficient, as it uses the best-known values for parameters; the users receive the computation log and the computation time lasts usually for several seconds. As mentioned in Sections II B and C, we do not compare these two methods one against the other; our goal is to classify the symmetric TSP instances used in this study and to report the results of two openly available and efficient software packages on them.

The performance on a given instance is measured using the *gap* adimensional parameter, given in formula (9):

$$gap = \frac{result - optimum}{optimum} \times 100 \tag{9}$$

This parameter is widely used in the literature [12] as it measures the quality results for all instances, no matter what values are stored in the distance matrices and the type of instances (structured, semi-structured, or unstructured). The value *optimum* in formula (9) is the optimum length (computed using Concorde as an exact solver for TSP) or the best-known solution if the optimum is not known.

For each instance, the ACO application executed ten trials and exited with the best (gap_min), average (gap_average), and worst (gap_max) values for the gap.

We reported the results on the three sets of instances: structured, semi-structured, and unstructured. For each instance, we computed the R index using formula (8), and the CV parameter specified in Section II.D using the formula (10).

$$CV = \frac{Std\_Dev(D)}{AVG(D)} \tag{10}$$

with D denoting the distance matrix. We also computed, for all instances, the correlation between every two parameters from *size*, CV, and R. We discussed the results for each data table.

As ACO has a solution at any time, to choose when to stop is the user's decision. We decided to execute 10 trials for each instance. The trial on structured and unstructured instances lasts for 60s. The *semi-structured* instances with less than 3000 nodes were solved for 60s, while the execution time for the larger ones was 300s.

The set of *structured* instances is composed of two subsets. The first subset includes 50 *Tnm* instances from [20]. These instances form a collection of 2D Euclidean instances with 52 to 199 vertices, distributed as orthogonal projections of a tetrahedron. The instances are described in Table 1. For each instance, we mention the number of vertices (*size*), the length of one optimum solution (taken from [20]), the CV and R values, computed by us.

The last two columns in Table 1 (CV and R) are both highly correlated with the second column (Size): *corr*(*size,* CV) is −0.858, *corr*(*size, R*) is −1.0, and *corr*(CV, R) is 0.858. The standard deviation for the values R is 0.16, while the standard deviation for CV values is 100 times smaller.

In Fig. 1 we have represented *Tnm100* together with one of its optimum solutions found by Concorde.

The second subset of structured instances contains new instances that we generated as follows. Based on the *NPeano* curve from [21] we devised 18 *Peano* instances extracted from a semi-regular tessellation with octahedrons and squares.

Fig. 2 (a) presents *peano9.9*, with nine octahedrons on each side. For generating other structured instances, we studied regular tessellations. We explored the possible polygons: equilateral triangle, square, or hexagon. We rejected the equilateral triangle as one of our restrictions was to consider integer coordinates for all the vertices. We also rejected the square, as a regular mesh has almost all the edges forming one optimum path of length 1 (similar to the Hamming distance). We generated four Hex instances from a regular tessellation with hexagons. The instance *hex9*, with a maximum of 9 hexagons stacked in a vertical direction, is shown in Fig. 2(b). We found the optimum solutions for all the new instances using Concorde. In Fig. 3 we present one optimum solution to *peano9.9* and *hex9*. The regularity of the optimum solution to *peano9* is remarkable, similar to a path in a 2D mesh. The optimum solution to *hex9* does not have a perfect pattern.

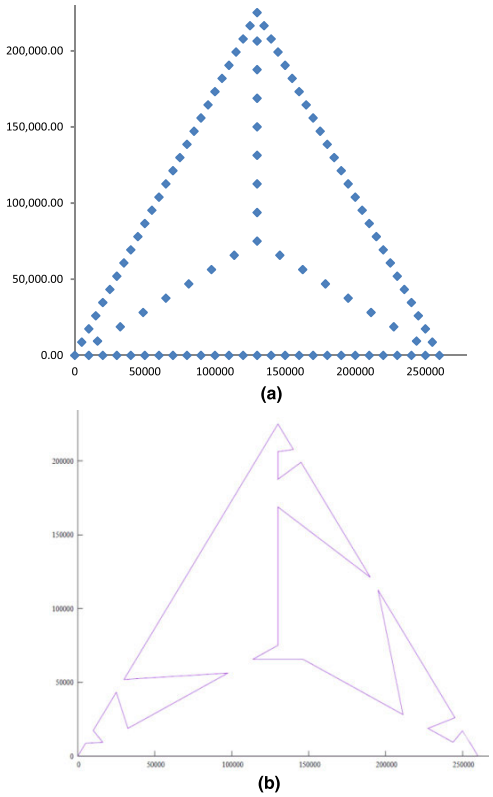


FIGURE 1. (a) *Tnm100* instance, vertices as blue dot. (b) One optimum solution as magenta line.

TABLE 2. Structured instances *Peano* and *Hex*.

Name	Size	Optimum	CV	R
peano3.3	72	108	0.484	1.061
peano4.4	128	192	0.482	1.029
peano5.5	200	300	0.480	1.010
peano6.6	288	432	0.479	0.998
peano7.7	392	588	0.478	0.990
peano8.8	512	768	0.477	0.984
peano9.9	648	972	0.477	0.979
peano12.12	1152	1728	0.476	0.970
peano15.15	1800	2700	0.476	0.964
peano18.18	2592	3888	0.476	0.961
peano21.21	3528	5292	0.476	0.958
peano27.18	3888	5832	0.499	0.958
peano24.24	4608	6912	0.476	0.956
peano27.27	5832	8748	0.476	0.955
peano30.30	7200	10800	0.476	0.953
peano36.27	7776	11664	0.488	0.953
peano33.33	8712	13068	0.476	0.952
peano36.36	10368	15552	0.476	0.952
hex3	24	25	0.510	1.512
hex5	54	56	0.488	1.401
hex7	96	98	0.480	1.352
hex9	150	152	0.478	1.325

Table 2 presents the characteristics of our new structured instances: size, length of one optimum solution, and the computed values for CV and R. The last two columns have again

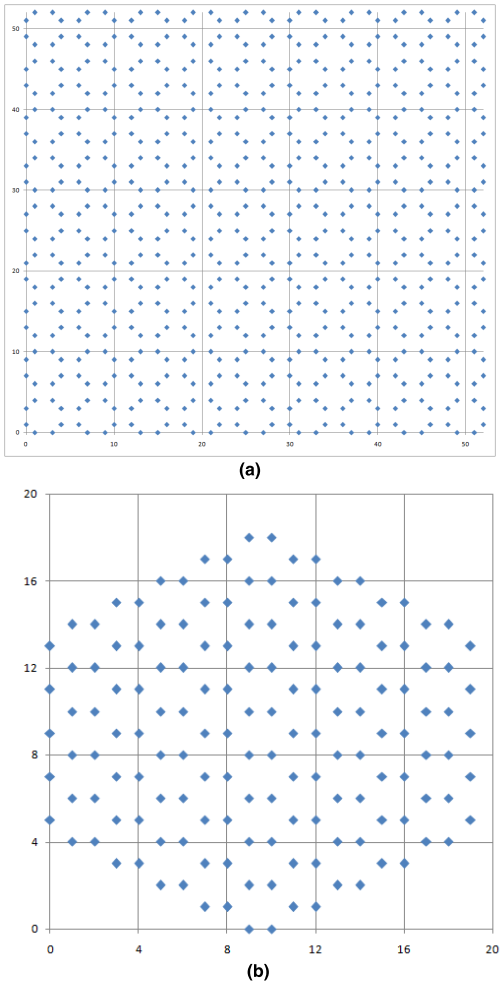


FIGURE 2. (a) *peano9.9* instance. (b) *hex9* instance.

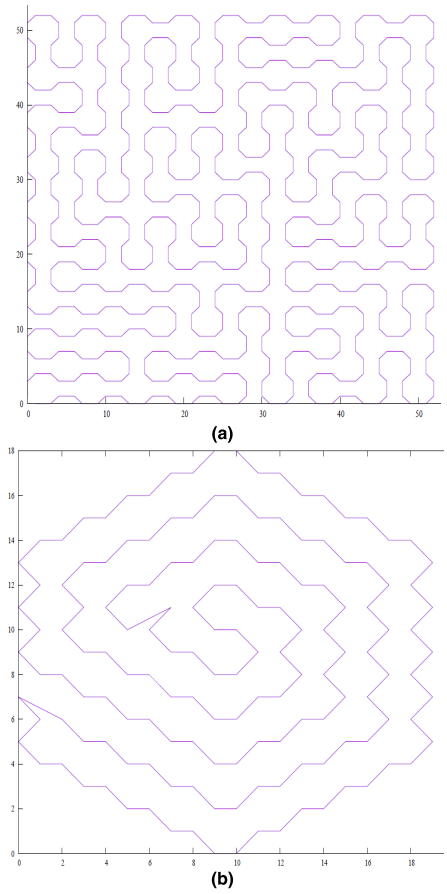


FIGURE 3. (a) One optimum solution to *peano9.9*. (b) One optimum solution to *hex9*.

a decreasing set of values, as the *size* increases: *corr*(*size,* CV) is −0.636, *corr*(*size,* R) is −0.974, and *corr*(CV, R) is 0.610. The standard deviation for the R index is 0.17, while the standard deviation for CV is 0.009.

Globally analyzed, these 72 structured instances are characterized by:

- *size* and R have an almost perfect inverse correlation;
- CV has a very low standard deviation of 0.01 and very slightly decreases when size increases;
- the standard deviation of R is 0.18, so R is more likely than CV to influence the results given by both ACO and LK methods.

The first subset of *semi-structured* instances contains 19 instances taken from [3]. These instances model the positions of the main localities from 19 countries; some of these instances are not solved to the optimum, yet. The other subset of semi-structured instances consists of 20 instances, each with *size*/100 clusters randomly spread in a 10^6 x 10^6 area. These instances were generated using the *portcgen* instance generator [75]. These last 20 instances were exactly solved using Concorde.

Table 3 presents the results for these instances. The R index shows that many instances from the first subset are close to being clustered (R < 0.3) or are semi-clustered (0.3 < R < 0.7).

For example, the settlements in *Egypt7146* instance (R = 0.275) are mainly on the banks of the Nile. The values CV and R do not correlate with the *size*: *corr*(CV, *size*) is 0.177, *corr*(*size,* R) is −0.250, and *corr*(CV, R) = −0.236. The instance measures *size*, CV, and R are therefore mutually independent for the *semi-structured* instances. These instances are treated separately in our following analysis. The standard deviation for CV is 0.1 and for R is 0.16. Unlike the *structured* and *random* instance sets, CV and R are spread in intervals with comparable lengths.

The *unstructured* instances are Random Uniform Euclidean (RUE) instances. Ten of them are new instances with random points with integer coordinates uniformly generated in the 1000 × 1000 square. Other 20 instances are generated using *portgen* [65], distributed in the $10^6 \times 10^6$ area.

Their characteristics are given in Table 4. All these instances were solved with Concorde for finding their optimum solutions.

The *rand1000* instance and one of its optimum solutions are presented in Fig. 4.

TABLE 3. ***Semi-structured*** **instances. The marked instances ∗ do not have a known optimum solution.**

Name	Size	Best known	CV	R
wi29	29	27603	0.605	0.890
dj38	38	6656	0.546	0.859
uy734	734	79114	0.481	0.797
zi929	929	95345	0.573	0.618
lu980	980	11340	0.497	0.459
rw1621	1621	26051	0.548	0.144
mu1979	1979	86891	0.863	0.279
nu3496	3496	96132	0.649	0.355
ca4663	4663	1290319	0.691	0.423
tz6117*	6117	*394718	0.520	0.649
eg7146*	7146	*172386	0.894	0.275
ym7663*	7663	*238314	0.751	0.410
pm8079*	8079	*114855	0.626	0.330
ei8246*	8246	*206171	0.494	0.785
ar9152*	9152	*837479	0.634	0.352
ja9847	9847	491924	0.683	0.298
gr9882	9882	300899	0.572	0.544
kz9976*	9976	*1061881	0.610	0.591
fi10639	10639	520527	0.532	0.647

Name	Size	Best known	CV	R
portc1000	1000	11387430	0.523	0.424
portc1100	1100	11882745	0.515	0.428
portc1200	1200	12609745	0.512	0.422
portc1300	1300	12890663	0.525	0.413
portc1400	1400	13106193	0.515	0.407
portc1500	1500	13497993	0.514	0.408
portc1600	1600	13820687	0.510	0.418
portc1700	1700	14147073	0.512	0.416
portc1800	1800	14822263	0.509	0.414
portc1900	1900	15007940	0.515	0.417
portc2000	2000	15376942	0.509	0.436
portc2100	2100	15976858	0.502	0.427
portc2200	2200	16214133	0.504	0.416
portc2300	2300	16575535	0.509	0.420
portc2400	2400	16690007	0.512	0.417
portc2500	2500	16908177	0.518	0.408
portc2600	2600	17172976	0.524	0.410
portc2700	2700	17594303	0.515	0.398
portc2800	2800	17736766	0.509	0.398
portc2900	2900	18406010	0.500	0.406

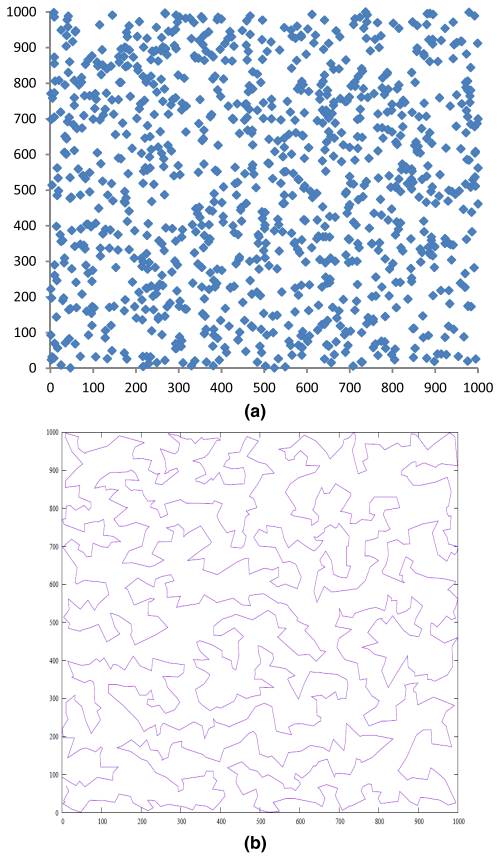


FIGURE 4. **(a)** ***rand1000*** **instance. (b)** ***One optimum solution to rand1000.***

The values for the R index are close to 1, corresponding to random instances according to classification based on R values, given in [9]. The values from the columns *size*, *CV* and *R* have the following correlations on pairs: *corr*(*size,* CV) is −0.641, *corr*(*size,* R) is −0.470, and *corr*(CV, R) = −0.345. The standard deviation for CV is 0.003, and the standard deviation for R is 0.02. The values in the CV column show a remarkable resemblance to the values from Table 1 and Table 2. The *CV* values belong to a more compact interval when compared to the domain of R.

Table 5 concludes the characterization of our sets and subsets of instances through the average values for CV and R. We can make the following observations. The average CV values for all the structured subsets are close to each other. The average obtained for the CV in our structured and unstructured sets of instances differs by less than 0.02. We decided to explore and compare the results on structured and unstructured instances in Section III.C. We observe that the average value for the R index on the semi-structured instances differentiates this class. Therefore, we performed a separate study on semi-structured instances in Section III.D.

B. EXPERIMENTAL RESULTS

Our experimental results reveal the characteristics of the solution obtained for the *structured*, *semi-structured,* and *unstructured* sets of instances when approached with the ACO and LK implementations used in this study.

All the *structured* instances were solved to optimality by ACO. LK solved to optimality only 31 out of the 72 structured instances. To analyze the outputs, we decided to extract from the computation logs, for the instances optimally solved both by ACO and LK, the number of needed iterations. These two samples are not mutually comparable, but we explore in Section III.C their individual connections with instance characteristics (*size*, CV, and R).

TABLE 4. Random TSP instances.

Name	Size	Optimum	CV	R
rand100	100	7671	0.490	1.052
rand200	200	10853	0.481	1.075
rand300	300	13021	0.483	1.009
rand400	400	14961	0.478	1.068
rand500	500	16405	0.478	1.017
rand600	600	17843	0.476	0.977
rand700	700	19698	0.476	1.047
rand800	800	20461	0.476	0.990
rand900	900	21856	0.476	1.007
rand1000	1000	23031	0.476	1.011
port1000	1000	23360648	0.476	1.035
port1100	1100	24368458	0.476	1.031
port1200	1200	25291021	0.476	1.035
port1300	1300	26336284	0.475	1.036
port1400	1400	27278945	0.476	1.028
port1500	1500	28227578	0.476	1.022
port1600	1600	29038070	0.475	1.017
port1700	1700	29767008	0.475	1.003
port1800	1800	30577201	0.475	1.000
port1900	1900	31507740	0.475	1.004
port2000	2000	32277771	0.475	1.007
port2100	2100	33103196	0.475	1.008
port2200	2200	33836367	0.475	1.003
port2300	2300	34595405	0.475	1.004
port2400	2400	35390280	0.475	1.005
port2500	2500	36059572	0.475	1.002
port2600	2600	36712641	0.475	0.999
port2700	2700	37389043	0.475	1.000
port2800	2800	38064813	0.475	1.002
port2900	2900	38703402	0.475	1.000

TABLE 5. Characterization of instance sets.

Set of instances	Average value for CV	Average value for R
Structured (Tables I and II)	0.494	0.947
Tnm (Table I)	0.500	0.901
Peano and Hex (Table II)	0.481	1.053
Semi-Structured (Table III)	0.565	0.462
Unstructured (Table IV)	0.477	1.016

Table 6 presents the instances features (*size*, CV, and R), the average iteration (in 10 trials) that found the reported solution for ACO (*aver_iter_ACO*), the iteration number when LK found its solution (*iter_LK*), and also the gaps obtained with ACO and LK implementations (*gap_ACO*, *gap_LK*). The execution time is 60 seconds for each ACO trial. If the non-zero values are less than 0.001, we wrote <0.001.

Table 7 presents the *semi-structured* instances features (*size*, CV, and R), the accuracy of the ACO approach (*gap_min*, *gap_average,* and *gap_max*), and also the accuracy obtained by LK implementation (*gap_LK*). Each ACO trial execution time is 60s for a *size* less than 3000, 300s otherwise.

Table 8 presents the *size*, CV, R, *gap_min*, *gap_average,* and *gap_max* values from ACO tests and *gap_LK* from the Lin-Kernighan method for the random instances. Like in the *structured* class, the ACO execution time is 60s for each trial.

Table 9 resumes the results presenting the averages for the three sets of data. To characterize the stability of the ACO method (details are given in Section III.C), we reported also the average of the difference between gap_max and gap_min). We performed a statistical investigation, using the *Analyse-it* add-in package, release 5.40.02. The analysis is oriented on the main groups of instances (structured, semi-structured, and random) and two subgroups of structured instances (*Hex* and *Peano*, and *Tnm*). The statistical investigation aims to underline the influence of the parameters (predictors) *size*, R, and CV on the behavior of the two algorithms considered in this article and on the quality of the solution. To achieve this goal we have computed:

- Statistics for *size*, R, and CV.
- Statistics for the values that measure the quality of the solution obtained using LK (*iter_LK, gap_LK*) and ACO (*aver_iter_ACO, gap_ACO*)
- Correlation between the parameters *iter_LK/ gap_LK/ aver_iter_ACO/ gap_ACO* and the predictors *size*, R and CV
- Regression models to capture the influence of the predictors *size*, R, and CV on the *iter_LK/ gap_LK/ aver_iter_ACO/ gap_ACO*. The models use a 95% confidence level for the estimates of the predictors *size*, R, and CV.

All the regression models and statistics are provided as supplementary materials to this paper and can be accessed on the web page [72], at the Section *Statistical analysis archive for the structured/semi-structured/random instances.*

In the next subsections, the predictors whose contribution is statistically significant for the model are specified for each class/subclass of instances.

C. ANALYSIS ON STRUCTURED AND UNSTRUCTURED INSTANCE CLASSES

In this subsection, we present the behavior of ACO and LK implementations on extreme sets of instances: *structured* and *unstructured*, which have several similar basic statistics. Tables 6 and 8 collect the results for the *structured* and the *randomly generated* instances. We note that we specially designed these two datasets such that to have as many as possible common features, but to be extreme from the point of view of the structural classification introduced by us. Therefore, the instances were chosen to have: almost the same average CV (0.494/0.477) and average R (0.947/1.016), compact CV values in the corresponding set, the same execution time (60s), the same number of trials (10), and the same number of artificial ants (25) for ACO. All the paired correlations show that the predictors: *size*, CV, and R are independent.

On *structured* instances, the ACO implementation managed to optimally solve all instances on any trial. There was

TABLE 6. Experimental results for the *structured* instances. ACO solved them to optimality, in all trials.

Name	Size	CV	R	aver_iter ACO	iter LK	gap ACO	gap LK
Tnm52	52	0.505	1.240	1.0	4	0	0
Tnm55	55	0.504	1.218	1.1	5	0	0
Tnm58	58	0.503	1.194	1.2	9	0	0.089
Tnm61	61	0.502	1.173	2.5	3	0	<0.001
Tnm64	64	0.502	1.153	1.0	59	0	0.260
Tnm67	67	0.501	1.135	1.0	43	0	0.112
Tnm70	70	0.501	1.119	5.7	40	0	<0.001
Tnm73	73	0.502	1.093	15.9	25	0	0.020
Tnm76	76	0.501	1.077	1.0	11	0	0.077
Tnm79	79	0.501	1.062	11.0	79	0	0
Tnm82	82	0.501	1.048	2.3	45	0	0
Tnm85	85	0.500	1.034	5.3	84	0	0.047
Tnm88	88	0.500	1.021	22.3	74	0	<0.001
Tnm91	91	0.500	1.009	3.4	89	0	0
Tnm94	94	0.500	0.997	14.8	83	0	0.016
Tnm97	97	0.500	0.980	10.7	94	0	<0.001
Tnm100	100	0.500	0.965	46.4	31	0	<0.001
Tnm103	103	0.500	0.955	14.3	51	0	0.039
Tnm106	106	0.499	0.944	15.9	88	0	0
Tnm109	109	0.499	0.930	185.8	62	0	0.007
Tnm112	112	0.499	0.917	274.5	50	0	0.042
Tnm115	115	0.499	0.905	361.9	41	0	<0.001
Tnm118	118	0.499	0.895	178.4	83	0	<0.001
Tnm121	121	0.499	0.881	446.4	92	0	0.013
Tnm124	124	0.499	0.870	918.6	95	0	<0.001
Tnm127	127	0.499	0.860	262.0	45	0	0.004
Tnm130	130	0.499	0.850	363.3	128	0	0.012
Tnm133	133	0.499	0.845	282.9	96	0	0.043
Tnm136	136	0.499	0.835	701.5	58	0	0.006
Tnm139	139	0.499	0.826	1092.6	120	0	<0.001
Tnm142	142	0.499	0.817	1531.9	14	0	0.003
Tnm145	145	0.499	0.809	373.3	137	0	0
Tnm148	148	0.499	0.800	943.1	60	0	0.008
Tnm151	151	0.499	0.792	1677.6	140	0	0
Tnm154	154	0.499	0.785	887.7	146	0	0.055
Tnm157	157	0.499	0.777	919.1	128	0	0.030
Tnm160	160	0.499	0.770	507.5	132	0	0.005
Tnm163	163	0.499	0.766	1489.1	142	0	0
Tnm166	166	0.499	0.759	1230.8	68	0	0.001
Tnm169	169	0.499	0.752	2433.4	138	0	<0.001
Tnm172	172	0.499	0.745	415.8	129	0	0.043
Tnm175	175	0.498	0.739	761.1	93	0	0.015
Tnm178	178	0.498	0.733	3714.5	173	0	0.038
Tnm181	181	0.498	0.727	4170.7	152	0	0.067
Tnm184	184	0.498	0.721	332.8	166	0	0.002
Tnm187	187	0.498	0.715	1164.2	16	0	0.006
Tnm190	190	0.498	0.710	789.6	114	0	0.016
Tnm193	193	0.498	0.706	2388.0	184	0	0.004
Tnm196	196	0.498	0.701	465.6	144	0	0.070
Tnm199	199	0.498	0.696	5175.5	33	0	0.001
peano3.3	72	0.484	1.061	1.0	0	0	0
peano4.4	128	0.482	1.029	1.0	0	0	0
peano5.5	200	0.480	1.010	2.4	0	0	0
peano6.6	288	0.479	0.998	8.8	0	0	0
peano7.7	392	0.478	0.990	16.8	0	0	0

TABLE 6. *(Continued.)* **Experimental results for the *structured* instances. ACO solved them to optimality, in all trials.**

peano8.8	512	0.477	0.984	23.5	30	0	0
peano9.9	648	0.477	0.979	26.4	0	0	0
peano12.12	1152	0.476	0.970	51.4	6	0	0
peano15.15	1800	0.476	0.964	94.8	17	0	0
peano18.18	2592	0.476	0.961	135.6	11	0	0
peano21.21	3528	0.476	0.958	161.6	117	0	0
peano27.18	3888	0.499	0.958	178.0	267	0	0
peano24.24	4608	0.476	0.956	194.9	592	0	0
peano27.27	5832	0.476	0.955	243.3	622	0	0
peano30.30	7200	0.476	0.953	261.4	1203	0	0
peano36.27	7776	0.488	0.953	275.7	1527	0	0
peano33.33	8712	0.476	0.952	287.1	329	0	0
peano36.36	10368	0.476	0.952	307.2	1485	0	0
hex3	24	0.510	1.512	1.0	0	0	0
hex5	54	0.488	1.401	1.2	0	0	0
hex7	96	0.480	1.352	12.8	0	0	0
hex9	150	0.478	1.325	25.4	62	0	0

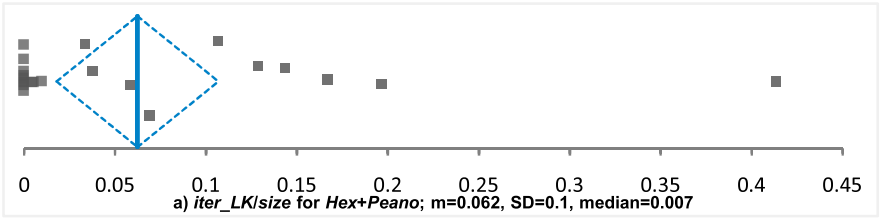


FIGURE 5. **Distribution and statistics of *iter_LK/size* for a) *Hex+Peano* instances and b) *Tnm* instances; confidence interval for the mean (as diamond): 95%.**

no need to perform 10 trials, as from the first one the optimum solution was delivered. This means that for similar structured instances, the computational resources can be saved by setting just one ACO trial per execution.

The LK implementation solved only 31 instances, producing an average $gap\_LK = 0.02$. To explore in-depth the difference between ACO and LK results, we studied the columns *aver_iter_ACO* and *iter_LK* from Table 6. These columns store the number of iterations of the corresponding application which found the solution.

The regression models (see Section III B) display the influence of CV and *size* on *iter_LK*. No relationship can be established with *size*, R, and/or CV for *gap_LK* and *aver_iter_ACO*. Although the predictors do not correlate, they are not enough to model the sample *gap_LK*. No predictions can be made for LK accuracy at the general level, for *structured* instances.

We are now refining the study on LK accuracy on the two structured subclasses: *Tnm* and *Hex+Peano*. The *Tnm* subclass was more difficult, although the average *size* is 126. The statistical model shows that the values of *size* barely influence *average iterations* in the case of ACO and not at all on *iter_LK*. The *gaps* returned by LK are not influenced by any predictor.

For the subclass *Hex+Peano*, the regression models reveal that *size* has a reduced influence on both *aver_iterations_ACO* and *iter_LK*, while R and CV do not influence at all. For this set of instances, LK performed equally better than ACO, as all *gap_LK* values were 0.

To assess the performance of LK on the structured subclasses *Hex+Peano* and *Tnm*, we used the ratio *iter_LK/size* as a measure of the earliness in finding a (good or even exact) solution. We explored these values as we have constant zero values for the *gap* for *Hex+Peano* instances. For *Hex+Peano*, LK is very efficient in terms of *iter_LK/size*, whose values are spread around the mean = 0.062 with SD = 0.1 (Fig. 5.a). For *Tnm*, the gaps are zero or very small (average value

TABLE 7. Experimental results for the *semi-structured* instances.

Name	Size	CV	R	gap_min ACO	gap_aver ACO	gap_max ACO	gap LK
wi29	29	0.605	0.890	0	0	0	0
dj38	38	0.546	0.859	0	0	0	0
uy734	734	0.481	0.797	0	0.019	0.061	0.188
zi929	929	0.573	0.618	0.066	0.117	0.172	0.111
lu980	980	0.497	0.459	0	0.118	0.212	0.088
rw1621	1621	0.548	0.144	0.031	0.143	0.380	0.230
mu1979	1979	0.863	0.279	0.579	0.836	1.122	0.154
nu3496	3496	0.649	0.355	0.716	1.439	2.042	0.536
ca4663	4663	0.691	0.423	1.059	1.309	1.506	0.153
tz6117*	6117	0.520	0.649	3.383	3.584	3.794	0.185
eg7146*	7146	0.894	0.275	2.739	3.071	3.363	0.074
ym7663*	7663	0.751	0.410	4.181	4.404	4.672	0.214
pm8079*	8079	0.626	0.330	4.143	4.366	4.452	0.786
ei8246*	8246	0.494	0.785	3.407	3.590	3.711	0.193
ar9152*	9152	0.634	0.352	3.839	4.065	4.280	0.361
ja9847	9847	0.683	0.298	4.142	4.590	4.853	0.165
gr9882	9882	0.572	0.544	4.710	5.131	5.395	0.567
kz9976*	9976	0.610	0.591	3.749	3.928	4.048	0.248
fi10639	10639	0.532	0.647	3.827	4.034	4.211	0.249
portc1000	1000	0.523	0.424	0.245	0.530	0.927	0.046
portc1100	1100	0.515	0.428	0.290	0.463	0.753	0
portc1200	1200	0.512	0.422	0.336	0.479	0.776	0.015
portc1300	1300	0.525	0.413	0.762	1.127	1.507	0.015
portc1400	1400	0.515	0.407	0.726	1.040	1.406	0.045
portc1500	1500	0.514	0.408	0.389	0.552	1.001	0.068
portc1600	1600	0.510	0.418	0.887	1.102	1.395	0.014
portc1700	1700	0.512	0.416	0.831	1.019	1.097	0.004
portc1800	1800	0.509	0.414	0.386	0.757	1.284	0.006
portc1900	1900	0.515	0.417	1.286	1.789	2.102	0.029
portc2000	2000	0.509	0.436	1.900	2.274	2.689	0.250
portc2100	2100	0.502	0.427	0.949	1.402	1.908	0.111
portc2200	2200	0.504	0.416	1.604	2.033	2.537	0.267
portc2300	2300	0.509	0.420	1.057	1.455	1.833	0.013
portc2400	2400	0.512	0.417	1.297	1.717	2.326	0.145
portc2500	2500	0.518	0.408	2.066	2.439	2.867	0.115
portc2600	2600	0.524	0.410	1.251	1.711	2.300	0.252
portc2700	2700	0.515	0.398	2.231	2.593	2.991	0.082
portc2800	2800	0.509	0.398	2.132	2.466	2.735	0.236
portc2900	2900	0.500	0.406	1.975	2.545	3.303	0.215

is 0.02) but the effort in finding the optimum or a very good approximation, measured with *iter_LK*/*size*, is ten times bigger than in the case of *Hex+Peano* (Fig. 5.b). We can conclude that the value of *iter_LK*/*size* correctly indicates the difficult *Tnm* instances.

We use the predictors' correlations to explain these differences inside the class of structured instances for LK. The correlations of predictor pairs show that they are highly correlated for the *Tnm* subclass. A strong negative correlation is between *size* and R, but CV does not correlate with *size* or R for the *Hex+Peano* instances. In our opinion, the high statistical correlation between the CV and R samples indicates that the class of *structured* instances is difficult for the LK method. This conjecture is worth being investigated for other sets of instances manifesting the same strong correlations between parameters.

On small unstructured instances, ACO implementation is highly sensitive to stochasticity: *Average (gap_max-gap_min)* = 0.26. Therefore multiple trials seem to bring high-quality benefits, balanced by spending more computing time.

When referring to *size*, we can observe that the ACO implementation optimally solved all the 10 *rand* instances.

The multiple regression analysis for ACO displays that both *size* and CV positively influence all ACO gaps: *gap_min, gap_average,* and *gap_max*. Higher values for *size* and/or CV determine higher gaps. Therefore, pre-computing CV could help in adjusting the number of trials and/or the stopping

TABLE 8. Experimental results for the *unstructured* instances.

Name	Size	CV	R	gap_min ACO	gap_aver ACO	gap_max ACO	gap LK
rand100	100	0.490	1.052	0	0	0	0
rand200	200	0.481	1.075	0	0	0	0
rand300	300	0.483	1.009	0	0	0	0
rand400	400	0.478	1.068	0	0	0	0
rand500	500	0.478	1.017	0	0	0	0.061
rand600	600	0.476	0.977	0	0	0	0
rand700	700	0.476	1.047	0	0.001	0.010	0.117
rand800	800	0.476	0.990	0	0.016	0.068	0.078
rand900	900	0.476	1.007	0	0.024	0.092	0.201
rand1000	1000	0.476	1.011	0	0.044	0.152	0.261
port1000	1000	0.476	1.035	0.075	0.119	0.189	0.166
port1100	1100	0.476	1.031	0.005	0.096	0.161	0.225
port1200	1200	0.476	1.035	0.004	0.061	0.179	0.146
port1300	1300	0.475	1.036	0.015	0.107	0.320	0.211
port1400	1400	0.476	1.028	0.079	0.208	0.365	0.065
port1500	1500	0.476	1.022	0.188	0.262	0.380	0.184
port1600	1600	0.475	1.017	0.182	0.305	0.428	0.376
port1700	1700	0.475	1.003	0.103	0.239	0.403	0.054
port1800	1800	0.475	1.000	0.182	0.267	0.345	0.161
port1900	1900	0.475	1.004	0.185	0.318	0.409	0.144
port2000	2000	0.475	1.007	0.113	0.387	0.509	0.134
port2100	2100	0.475	1.008	0.270	0.478	0.725	0.197
port2200	2200	0.475	1.003	0.519	0.701	0.941	0.136
port2300	2300	0.475	1.004	0.595	0.992	1.400	0.118
port2400	2400	0.475	1.005	0.317	0.418	0.624	0.096
port2500	2500	0.475	1.002	0.463	0.760	0.951	0.206
port2600	2600	0.475	0.999	0.552	0.777	1.029	0.161
port2700	2700	0.475	1.000	0.492	0.791	1.364	0.103
port2800	2800	0.475	1.002	0.771	0.968	1.323	0.157
port2900	2900	0.475	1.000	0.846	1.093	1.376	0.144

TABLE 9. Average values for *structured*, *semi-structured* and *unstructured* instances.

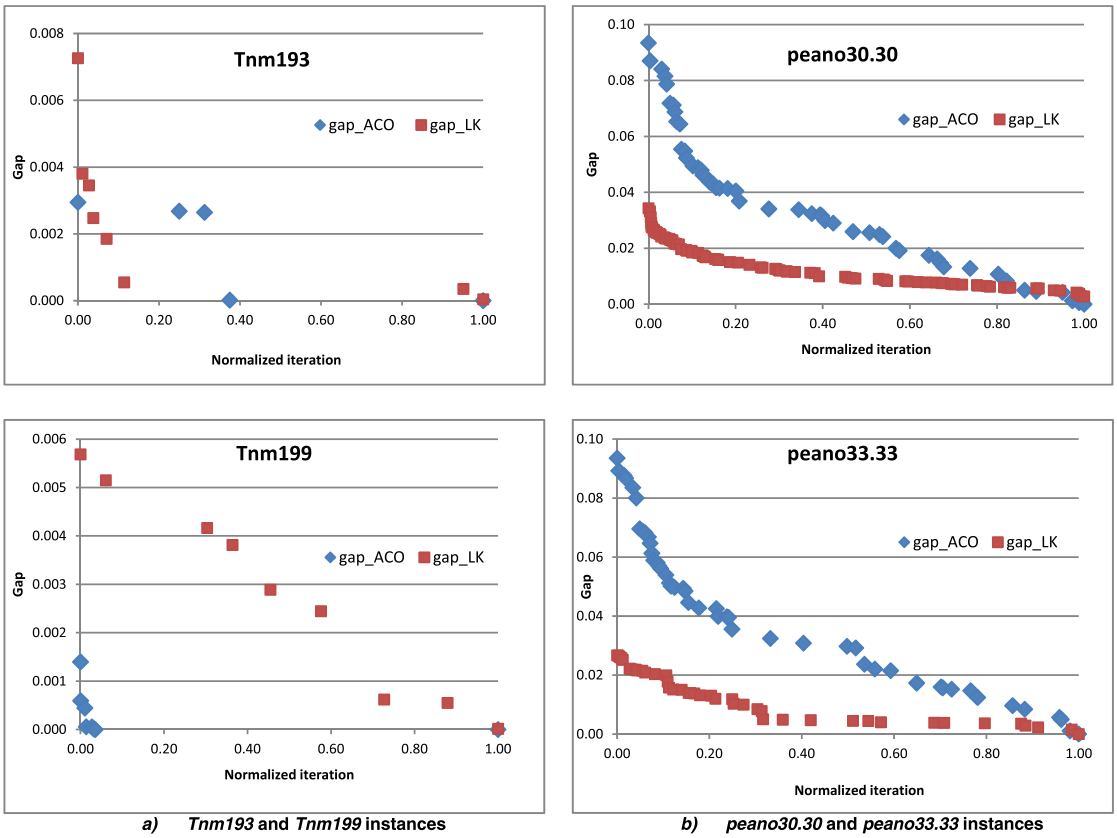
Set of instances	Aver. Size	Aver. CV	Aver. R	Aver. gap_min ACO	Aver. gap_aver ACO	Aver. gap_max ACO	Aver.gap_max-gap_min ACO	Aver. gap_LK
Structured	**921**	**0.494**	**0.947**	**0**	**0**	**0**	**0**	**0.02**
Tmn	126	0.500	0.901	0	0	0	0	0.023
Peano	3316	0.479	0.977	0	0	0	0	0
Hex	81	0.489	1.397	0	0	0	0	0
Semi-structured	**3595**	**0.565**	**0.462**	**1.48**	**1.77**	**2.10**	**0.52**	**0.17**
Unstructured	**1483**	**0.477**	**1.016**	**0.20**	**0.31**	**0.46**	**0.26**	**0.13**

condition for ACO. The *gap_LK* sample is not predicted by any combination of *size*, R, or CV.

We switch our analysis to methods' evolution in time. We use the fitness landscape ideas to observe and to discuss the sequence of the solutions provided by ACO and LK implementations. For ACO we chose one trial. Each method provides a sequence of increasingly better solutions. Therefore we define an ad-hoc neighborhood formed by the two neighbors in this sequence, and as the fitness function, we consider the *gap* already defined. We normalized the iterations, as our goal was to represent in the same chart the fitness landscapes of both solving methods. We present four structured instances in Fig. 6 and four random instances in Fig. 7, all with higher *size* in their sub-classes. We also counted in Table 10 how many intermediate solutions (*steps*) were found by each method.

The *Tnm* instances represent again an outlier class. Although both methods have very short sequences of consecutive solutions (very few steps in Table 10), ACO rapidly finds the optimum solution, while LK struggles and in most of the cases fails to find the optimum.

The landscape for the other six instances displays a common aspect: ACO starts from a higher *gap* but has a steeper descent than LK. *Port* instances are more difficult for ACO; LK is very effective at the beginning, so ACO is not able to overcome the LK excellent start. ACO has also less improved



a) ***Tnm193*** **and** ***Tnm199*** **instances** *b)* ***peano30.30*** **and** ***peano33.33*** **instances**

FIGURE 6. **Evolution of *gap_ACO* and *gap_LK* for a) *Tnm193* and *Tnm199* instances and b) *peano30.30* and *peano33.33* instances.**

solutions; in this case, showing that LK maintains its capacity to find better solutions, even their improvement slows down.

To conclude, the structural characteristics have an important impact on the behavior of both implementations. Parameters like CV or R do not always offer enough information on how difficult an instance is for a given solving method. ACO perfectly solved the structured instances from our experiments but seems to not have the same performances on large unstructured ones. To classify an instance correctly is therefore helpful in choosing the most effective heuristic method.

We also pointed out that the classification of an instance as structured is not enough for making predictions for LK accuracy. In the case of *Tnm* instances, the three instance parameters: size, CV, and R highly correlate, so they are not able to capture more than one feature. In our opinion, this is why the LK accuracy cannot be predicted.

The fitness landscape of these two classes is similar, with the already discussed exception of the *Tnm* instances. If these extreme instances are not considered, the structured and random classes are not separated from this point of view.

D. ANALYSIS ON SEMI-STRUCTURED INSTANCES

This subsection is dedicated to discussions on the *semi-structured* instances.

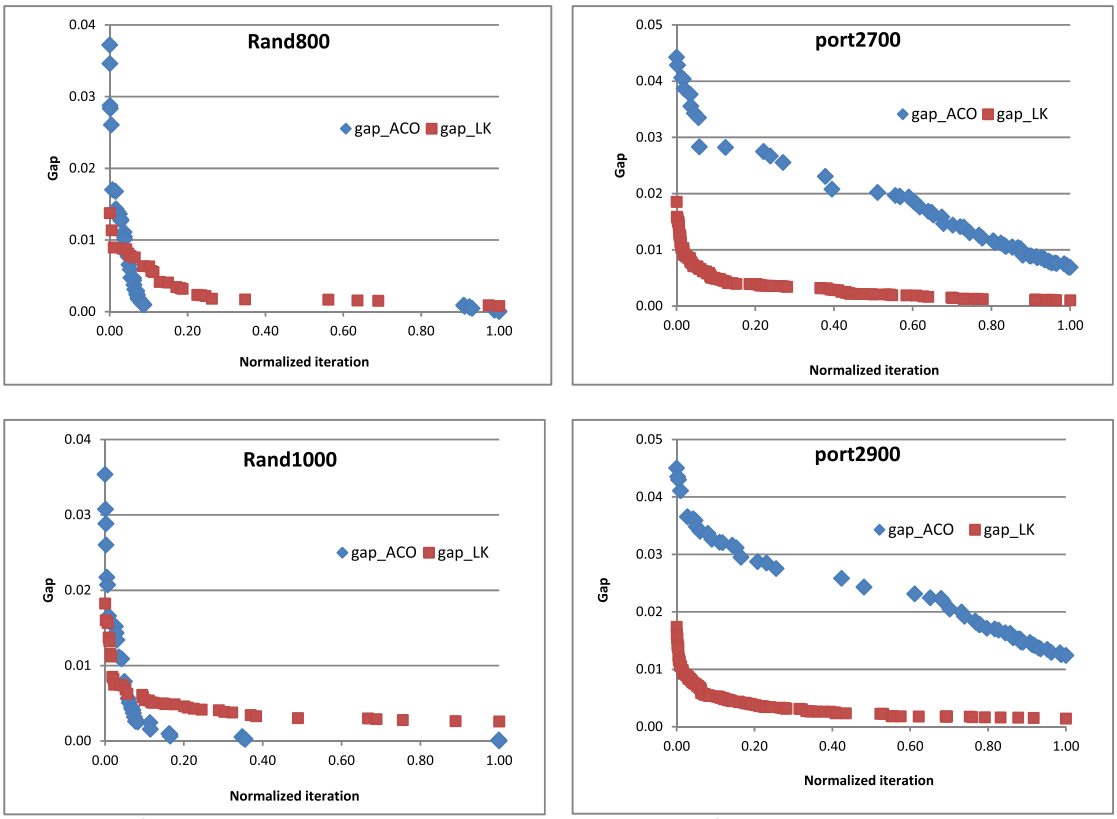
The class of semi-structured instances is presented in Table 7. The set of ACO solutions in 10 trials had a large standard deviation. Intensive computations are therefore needed for obtaining good solutions with ACO.

TABLE 10. **Number of steps for *structured* and *unstructured* instances.**

Type	Instances	steps ACO	steps LK
Structured	Tnm193	5	8
	Tnm199	8	9
	peano30.30	52	90
	peano33.33	48	42
Unstructured	Rand800	45	27
	Rand1000	33	38
	port2700	63	118
	port2900	53	125

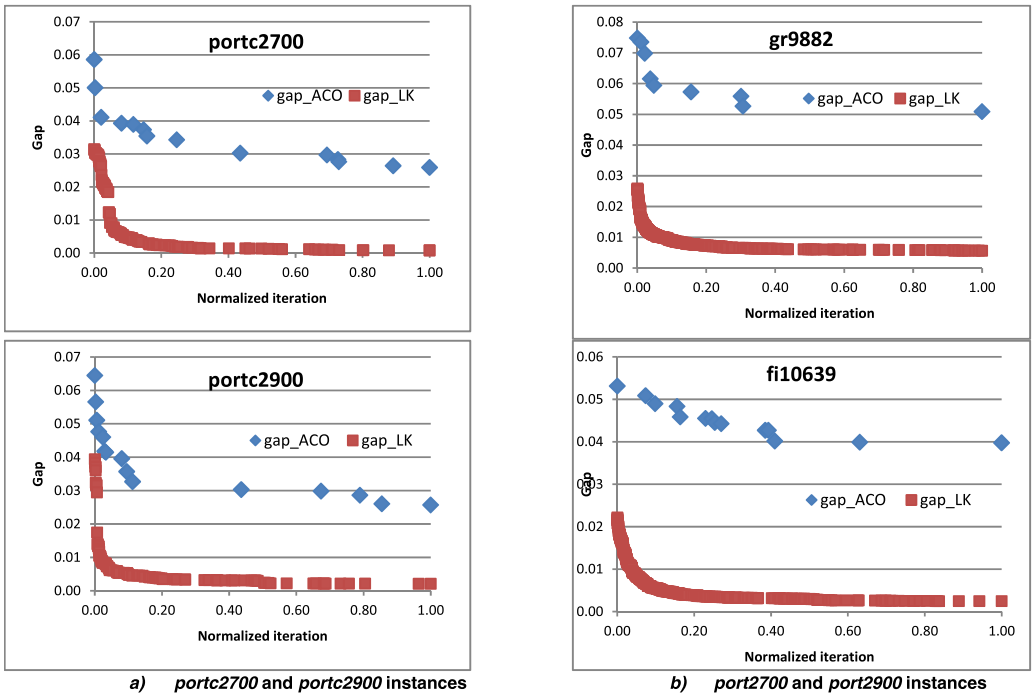
For the class of semi-structured instances, the regression models show that the parameters *size* and CV influence *gap_min*, *gap_average,* and *gap_max*, while R has an impact on *gap_average* and *gap_max*. This means that if a large class of semi-structured instances must be solved, Machine Learning techniques could be used for gap prediction. None of the predictors *size*, R, and CV does have significance on *gap_LK*. Although LK shows good accuracy, the lack of a reliable and simple equation for predicting the gaps means that all the instances in a set must be solved.

ACO and LK had contrasting results on the 20 clustered random instances from the *semi-structured* subset versus the



a) *Rand800* and *Rand1000* instances b) *port2700* and *port2900* instances

FIGURE 7. Evolution of *gap_ACO* and *gap_LK* for a) *Rand800* and *Rand1000* instances and b) *port2700* and *port2900* instances.



a) *portc2700* and *portc2900* instances b) *port2700* and *port2900* instances

FIGURE 8. Evolution of *gap_ACO* and *gap_LK* for a) *Rand800* and *Rand1000* instances and b) *port2700* and *port2900* instances.

20 *unstructured* random instances. The clustered instances were more difficult for ACO than the random subset. LK manifested the opposite behavior.

Fig. 8 shows the fitness landscapes for four semi-structured instances. The number of improvements made by ACO and LK is presented in Table 11.

TABLE 11. Number of steps for *structured* and *unstructured* instances.

Type	Instances	steps ACO	steps LK
Semi-structured	portc2700	14	143
	portc2900	15	115
	gr9882	9	376
	fi10639	14	398

These instances clearly separate from the other two classes from the ACO-behavior point of view. ACO has a poorer start and is not able to recover later. Not only the gaps are improving slowly, but the steps are very few when compared to LK. LK maintains the same landscape evolution: steep at the beginning, slower at its final.

IV. DISCUSSIONS AND CONCLUSIONS

This work empirically studies the quality of the results provided by open implementations of Ant Colony Optimization (ACO) and Lin-Kernighan (LK) when 2D Euclidean TSP instances with different structures are approached. Knowing the impact of specific characteristics of the instance on the quality of the result could orient the decision on what solver to choose when several solvers are available. Also, when a set of similar instances has to be solved, trustworthy predictions could orient the user in setting application parameters.

The results show that *structured* instances, with a regular distribution of vertices or with tendencies towards clustering, are usually easier for ACO than random instances with the same *size* and same CV parameter. Also, for *structured* instances, ACO is very stable so fewer trials are needed, which lowers the computation time. For the *semi-structured* group of instances, we show that good predictions can be made for ACO *gaps* based on two parameters. The LK implementation used in our work optimally solved the *Hex* and *Peano* instances.

From the landscape point of view, ACO was sensitive to the instance structure, while LK had a uniform behavior.

It is important to emphasize that our experiment considered two representative metaheuristics, one from the problem–independent heuristics category (ACO) and the other from the local search heuristics group (LK), without comparing them against each other or stating their supremacy over other metaheuristics.

The main goals of the paper were to prove that the structure of the TSP instance strongly influences the quality of its solution, to propose and empirically analyze a structure-based classification of TSP, and to design several TSP instances with *apriori*-known characteristics. We did not intend to compare the considered methods from both the point of view of computational cost and closeness of the provided solution to the optimum, but only to indicate to the readers the benefit they can have using additional information about the structure of the instances when choosing the solver.

We will extend this study by using several metaheuristics with known promising results for optimization problems in general and symmetric TSP in particular. Some of these metaheuristics are mentioned in Section II.A: Discrete Monarch Optimization [46], [50], African Buffalo Optimization [51], Earthworm optimization [47], Elephant Herding Optimization [48], [49], Moth Search [80].

The instances generated for this work have *a priori* specified features. These datasets could be used for further investigations by the research community. Their design could also be used to extend these collections with higher *size* instances or to include other complementary features.

To the best of our knowledge, results concerning human performance on the *Tnm* set are not yet reported in the literature. *Tnm* instances have many optimal solutions, so we suppose that they are not difficult for humans. As a further direction of our study, we intend to search for a method for generating small-size TSP instances that are efficiently solved by humans but are difficult for computers.

REFERENCES

[1] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA, USA: MIT Press, 2004.

[2] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Oper. Res.*, vol. 21, no. 2, pp. 498–516, Apr. 1973, doi: 10.1287/opre.21.2.498.

[3] *National Traveling Salesman Problems*. Accessed: Mar. 1, 2020. [Online]. Available: http://www.math.uwaterloo.ca/tsp/world/countries.html

[4] S. Thiébaux, J. Slaney, and P. Kilby, "Estimating the hardness of optimization," in *Proc. 14th Eur. Conf. Artif. Intell. (ECAI)*, 2000, pp. 123–127, doi: 10.5555/3006433.3006460.

[5] W. J. Cook, *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton, NJ, USA: Princeton Univ. Press, 2012.

[6] S. Arora, "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems," *J. ACM*, vol. 45, no. 5, pp. 753–782, Sep. 1998, doi: 10.1145/290179.290180.

[7] P. Cheeseman, B. Kanefsky, and W. M. Taylor, "Where the really hard problems are," in *Proc. 12th Int. Joint Conf. Artif. Intell. IJCAI*, San Mateo, CA, USA: Morgan Kaufmann, vol. 1, Aug. 1991, pp. 331–337, doi: 10.5555/1631171.1631221.

[8] A. Mariano, P. Moscato, and M. G. Norman, "Using L-systems to generate arbitrarily large instances of the Euclidean traveling salesman problem with known optimal tours," in *Proc. Anales del 27th Simposio Brasileiro de Pesquisa Operacional*, 1995, pp. 6–8.

[9] M. Dry, K. Preiss, and J. Wagemans, "Clustering, randomness, and regularity: Spatial distributions and human performance on the traveling salesperson problem and minimum spanning tree problem," *J. Problem Solving*, vol. 4, no. 1, pp. 1–17, Feb. 2012, doi: 10.7771/1932-6246.1117.

[10] J. N. MacGregor, "Effects of cluster location on human performance on the traveling salesperson problem," *J. Problem Solving*, vol. 5, no. 2, pp. 33–41, Apr. 2013, doi: 10.7771/1932-6246.1151.

[11] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artif. Intell.*, vol. 26, pp. 79–111, Jan. 2014.

[12] P. Kerschke, L. Kotthoff, J. Bossek, H. H. Hoos, and H. Trautmann, "Leveraging TSP solver complementarity through machine learning," *Evol. Comput.*, vol. 26, no. 4, pp. 597–620, Dec. 2018, doi: 10.1162/evco_a_00215.

[13] J. Pihera and N. Musliu, "Application of machine learning to algorithm selection for TSP," in *Proc. IEEE 26th Int. Conf. Tools with Artif. Intell.*, Nov. 2014, pp. 47–54, doi: 10.1109/ICTAI.2014.18.

[14] K. J. S.-M. van Hemert and X. Y. Lim, "Understanding TSP difficulty by learning from evolved instances," in *Learning and Intelligent Optimization* (Lecture Notes in Computer Science), vol. 6073, C. Blum and R. Battiti, Eds. Berlin, Germany: Springer, 2010.

[15] K. Helsgaun, "General k-opt submoves for the Lin–Kernighan TSP heuristic," *Math. Program. Comput.*, vol. 1, nos. 2–3, pp. 119–163, Oct. 2009.

[16] X.-F. Xie and J. Liu, "Multiagent optimization system for solving the traveling salesman problem (TSP)," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 39, no. 2, pp. 489–502, Apr. 2009, doi: 10.1109/TSMCB.2008.2006910.

[17] D. Sanches, D. Whitley, and R. Tinós, "Building a better heuristic for the traveling salesman problem," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2017, pp. 329–336, doi: 10.1145/3071178.3071305.
[18] L. Kotthoff, "Algorithm selection for combinatorial search problems: A survey," *AI Mag.*, vol. 35, no. 3, pp. 48–60, Sep. 2014.
[19] K. Smith-Miles and J. van Hemert, "Discovering the suitability of optimisation algorithms by learning from evolved instances," *Ann. Math. Artif. Intell.*, vol. 61, no. 2, pp. 87–104, Feb. 2011.
[20] S. Hougardy and X. Zhong, "Hard to solve instances of the Euclidean traveling salesman problem," 2018, *arXiv:1808.02859*. [Online]. Available: http://arxiv.org/abs/1808.02859
[21] M. G. Norman and P. Moscato, "The Euclidean traveling salesman problem and a space-filling curve," *Chaos Soliton Fract*, vol. 6, pp. 389–397, Jan. 1995, doi: 10.1016/0960-0779(95)80046-J.
[22] G. B. Dantzig, *Linear Programming and Extensions*. Princeton, NJ, USA: Princeton Univ. Press, 1963.
[23] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *J. Oper. Res. Soc. Amer.*, vol. 2, no. 4, pp. 393–410, Nov. 1954.
[24] T. Arthanari and K. Qian, "Symmetric travelling salesman problem," in *Mathematical Programming and Game Theory* (Indian Statistical Institute Series), S. Neogy, R. Bapat, and D. Dubey, Eds. Singapore: Springer, 2018, pp. 87–114, doi: 10.1007/978-981-13-3059-9_5.
[25] R. Roberti and P. Toth, "Models and algorithms for the asymmetric traveling salesman problem: An experimental comparison," *EURO J. Transp. Logistics*, vol. 1, nos. 1–2, pp. 113–133, Jun. 2012, doi: 10.1007/s13676-012-0010-0.
[26] M. Bläser, "Metric TSP," in *Encyclopedia of Algorithms*, M. Y. Kao, Ed. Boston, MA, USA: Springer, 2008.
[27] A. Czumaj, "Euclidean traveling salesman problem," in *Encyclopedia of Algorithms*, M. Y. Kao, Ed. New York, NY, USA: Springer, 2016.
[28] G. Mosheiov, "The travelling salesman problem with pick-up and delivery," *Eur. J. Oper. Res.*, vol. 79, no. 2, pp. 299–310, Dec. 1994.
[29] K. Ilavarasi and K. S. Joseph, "Variants of travelling salesman problem: A survey," in *Proc. Int. Conf. Inf. Commun. Embedded Syst. (ICICES)*, Feb. 2014, pp. 1–7, doi: 10.1109/ICICES.2014.7033850.
[30] N. Agatz, P. Bouman, and M. Schmidt, "Optimization approaches for the traveling salesman problem with drone," *Transp. Sci.*, vol. 52, no. 4, pp. 965–981, Aug. 2018, doi: 10.1287/trsc.2017.0791.
[31] G. C. Onwubolu and M. Clerc, "Optimal path for automated drilling operations by a new heuristic approach using particle swarm optimization," *Int. J. Prod. Res.*, vol. 42, no. 3, pp. 473–491, Feb. 2004, doi: 10.1080/00207540310001614150.
[32] O. Johnson and J. Liu, "A traveling salesman approach for predicting protein functions," *Source Code Biol. Med.*, vol. 1, no. 1, p. 3, 2006, doi: 10.1186/1751-0473-1-3.
[33] E. Bonabeau, F. Henaux, S. Guérin, D. Snyer, P. Kuntz, and G. Theraulaz, "Routing in telecommunications networks with ant-like agents," in *Intelligent Agents for Telecommunication Applications* (Lecture Notes in Computer Science), vol. 1437, S. Albayrak and F. J. Garijo, Eds. Berlin, Germany: Springer, 1998, pp. 60–71, doi: 10.1007/BFb0053944.
[34] P. Jaillet, "Probabilistic travelling salesman problems," Ph.D. dissertation, Dept. Civil Eng., Cambridge, MA, USA: MIT Press, 1985.
[35] G. C. Crisan and E. Nechita, "Solving fuzzy TSP with ant algorithms," *Int. J. Comput., Commun. Control*, vol. 3, pp. 228–231, Jan. 2008.
[36] Z. Wang, J. Guo, M. Zheng, and Y. Wang, "Uncertain multiobjective traveling salesman problem," *Eur. J. Oper. Res.*, vol. 241, no. 2, pp. 478–489, Mar. 2015, doi: 10.1016/j.ejor.2014.09.012.
[37] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. New York, NY, USA: Plenum Press, 1972, pp. 85–103.
[38] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem Concorde TSP Solver*. Accessed: Mar. 1, 2020. [Online]. Available: http://www.math.uwaterloo.ca/tsp/concorde/
[39] *NEOS Interfaces to Concorde*. Accessed: Mar. 1, 2020. [Online]. Available: https://neos-server.org/neos/solvers/co:concorde/TSP.html
[40] H. A. Abdulkarim and I. F. Alshammari, "Comparison of algorithms for solving traveling salesman problem," *Int. J. Eng. Adv. Tech.*, vol. 4, no. 6, pp. 76–79, Aug. 2015.
[41] Y. A. Tan, X. H. Zhang, L. N. Xing, X. L. Zhang, and S. W. Wang, "An improved multi-agent approach for solving large traveling salesman problem," in *Agent Computing and Multi-Agent Systems* (Lecture Notes in Computer Science), vol. 4088, Z. Z. Shi and R. Sadananda, Eds. Berlin, Germany: Springer, 2006, pp. 351–362.
[42] M. M. Abid and M. Iqbal, "Heuristic approaches to solve traveling salesman problem," *TELKOMNIKA Indonesian J. Elect. Eng.*, vol. 15, no. 2, pp. 390–396, 2015, doi: 10.11591/telkomnika.v15i2.8301.
[43] G. C. Crişan, E. Nechita, and V. Palade, "On the effect of adding nodes to TSP instances: An empirical analysis," in *Advances in Combining Intelligent Methods* (Intelligent Systems Reference Library), vol. 116, I. Hatzilygeroudis, V. Palade, and J. Prentzas, Eds. Cham, Switzerland: Springer, 2017, pp. 25–45, doi: 10.1007/978-3-319-46200-4_2.
[44] Y. Marinakis, "Heuristic and metaheuristic algorithms for the traveling salesman problem," in *Encyclopedia of Optimization*, C. A Floudas, P. M. Pardalos, Eds. Boston, MA, USA: Springer, 2009, pp. 1498–1506, doi: 10.1007/978-0-387-74759-0_262.
[45] D. Gupta, "Solving TSP using various Meta-Heuristic algorithms," *Int. J. Recent Contribution From Eng., Sci. IT*, vol. 1, no. 2, pp. 22–26, 2013.
[46] G.-G. Wang, S. Deb, and Z. Cui, "Monarch butterfly optimization," *Neural Comput. Appl.*, vol. 31, no. 7, pp. 1995–2014, Jul. 2019, doi: 10.1007/s00521-015-1923-y.
[47] G. G. Wang, S. Deb, and L. S. Coelho, "Earthworm optimisation algorithm: A bio-inspired metaheuristic algorithm for global optimisation problems," *Int. J. Bio-Inspired Comput.*, vol. 12, no. 1, pp. 1–22, 2018, doi: 10.1504/IJBIC.2015.10004283.
[48] G.-G. Wang, S. Deb, and L. D. S. Coelho, "Elephant herding optimization," in *Proc. 3rd Int. Symp. Comput. Bus. Intell. (ISCBI)*, Dec. 2015, pp. 1–5, doi: 10.1109/ISCBI.2015.8.
[49] A. Hossam, A. Bouzidi, and M. E. Riffi, "Elephants herding optimization for solving the travelling salesman problem," in *Advanced Intelligent Systems for Sustainable Development* (Advances in Intelligent Systems and Computing), vol. 912, M. Ezziyyani, Ed. Cham, Switzerland: Springer, 2019, pp. 122–130, doi: 10.1007/978-3-030-12065-8_12.
[50] G. G. Wang, G. S. Hao, S. Cheng, and Q. Q. Qin, "A discrete monarch butterfly optimization for Chinese TSP problem," in *Advances in Swarm Intelligence* (Lecture Notes in Computer Science), vol. 9712, Y. Tan, Y. Shi, B. Niu, Eds. Cham, Switzerland: Springer, 2016, pp. 165–173, doi: 10.1007/978-3-319-41000-5_16.
[51] J. B. Odili and M. N. Mohmad Kahar, "Solving the traveling Salesman's problem using the african buffalo optimization," *Comput. Intell. Neurosci.*, vol. 2016, pp. 1–12, Jan. 2016, doi: 10.1155/2016/1510256.
[52] D. Whitley and W. Chen, "Constant time steepest descent local search with lookahead for NK-landscapes and MAX-kSAT," in *Proc. 14th Int. Conf. Genetic Evol. Comput. Conf. GECCO*, Jul. 2012, pp. 1357–1364.
[53] M. Guntsch and M. Middendorf, "Applying population based ACO to dynamic optimization problems," in *Proc. Int. Workshop Ant Algorithms*, vol. 2463, Brussels, Belgium, Sep. 2002, pp. 111–122.
[54] A. Maity and S. Das, "Efficient hybrid local search heuristics for solving the travelling thief problem," *Appl. Soft Comput.*, vol. 93, Aug. 2020, Art. no. 106284, doi: 10.1016/j.asoc.2020.106284.
[55] C. Blum, "Ant colony optimization: Introduction and recent trends," *Phys. Life Rev.*, vol. 2, no. 4, pp. 353–373, 2005, doi: 10.1016/j.plrev.2005.10.001.
[56] J. Brownlee. (2011). *Clever Algorithms: Nature-Inspired Programming Recipes*. Accessed: Mar. 22, 2020. [Online]. Available: http://www.cleveralgorithms.com/
[57] M. Dorigo and T. Stützle, "The ant colony optimization metaheuristic: Algorithms, applications, and advances," in *Handbook of Metaheuristics* (International Series in Operations Research & Management Science), vol. 57, F. Glover, G. A. Kochenberger, and G. A., Eds. Boston, MA, USA: Springer, 2003, doi: 10.1007/0-306-48056-5_9.
[58] Y. M. Yue and X. Wang, "An improved ant colony optimization algorithm for solving TSP," *Int. J. Multimedia Ubiquitous Eng.*, vol. 10, no. 12, pp. 153–164, Dec. 2015, doi: 10.14257/ijmue.2015.10.12.16.
[59] R. Skinderowicz, "Ant colony system with a restart procedure for TSP," in *Computational Collective Intelligence. ICCCI* (Lecture Notes in Computer Science), vol. 9876, N. Nguyen, L. Iliadis, Y. Manolopoulos, and B. Trawiński, Eds. Cham, Switzerland: Springer, 2016, doi: 10.1007/978-3-319-45246-3_9.
[60] M. Kurdi, "An ant colony system with a novel non-DaemonActions procedure for multiprocessor task scheduling in multistage hybrid flow shop," *Swarm Evol. Comput.*, vol. 44, pp. 987–1002, Feb. 2019, doi: 10.1016/j.swevo.2018.10.012.
[61] H. Ismkhan, "Effective heuristics for ant colony optimization to handle large-scale problems," *Swarm Evol. Comput.*, vol. 32, pp. 140–149, Feb. 2017, doi: 10.1016/j.swevo.2016.06.006.

[62] M. López-Ibáñez and T. Stützle. *ACOTSPQAP: Ant Colony Optimization Algorithms for the Travelling Salesman Problem and the Quadratic Assignment Problem*. Accessed: Mar. 1, 2020. [Online]. Available: http://iridia.ulb.ac.be/aco-tsp-qap/
[63] D. S. Johnson and L. A. McGeoch, "Local search in combinatorial optimization," in *The Traveling Salesman Problem: A Case Study in Local Optimization*. New York, NY, USA: Wiley, 1996.
[64] O. Martin, S. W. Otto, and E. W. Felten, "Large-step Markov chains for the traveling salesman problem," *J. Complex Syst.*, vol. 1991, vol. 5, no. 3, p. 299.
[65] C. H. Papadimitriou, "The Euclidean traveling salesman problem is NP-complete," *Theor Comput Sci*, vol. 4, no. 3, pp. 237–244, 1977, doi: 10.1016/0304-3975(77)90012-3.
[66] O. Mersmann, B. Bischl, J. Bossek, H. Trautmann, M. Wagner, and F. Neumann, "Local search and the traveling salesman problem: A feature-based characterization of problem hardness," in *Proc. 6th Int. Conf. Learn. Intell. Optim.*, Y. Hamadi and M. Schoenauer, Eds. Berlin, Germany: Springer, 2012, pp. 115–129, doi: 10.1007/978-3-642-34413-8_9.
[67] O. Mersmann, B. Bischl, H. Trautmann, M. Wagner, J. Bossek, and F. Neumann, "A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem," *Ann. Math. Artif. Intell.*, vol. 69, no. 2, pp. 151–182, Oct. 2013, doi: 10.1007/s10472-013-9341-2.
[68] S. Nallaperuma, M. Wagner, and F. Neumann, "Analyzing the effects of instance features and algorithm parameters for max–min ant system and the traveling salesperson problem," *Frontiers Robot. AI*, vol. 2, p. 18, Jul. 2015, doi: 10.3389/frobt.2015.00018.
[69] J. Bossek, P. Kerschke, A. Neumann, M. Wagner, F. Neumann, and H. Trautmann, "Evolving diverse TSP instances by means of novel and creative mutation operators," in *Proc. 15th ACM/SIGEVO Conf. Found. Genetic Algorithms - FOGA*, 2019, pp. 58–71, doi: 10.1145/3299904.3340307.
[70] *TSPLIB*. Accessed: Mar. 1, 2020. [Online]. Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
[71] *TSP Test Data*. Accessed: Mar. 1, 2020. [Online]. Available: http://www.math.uwaterloo.ca/ tsp/data/index.html
[72] *2D Euclidean TSP Instances*. Accessed: Mar. 28, 2020. [Online]. Available: http://cadredidactice.ub.ro/ceraselacrisan/2d-euclidean-tsp-instances/
[73] G. C. Crişan, C. M. Pintea, P. C. Pop, and O. Matei, "Economical connections between several European countries based on TSP data," *Log. J. IGPL*, vol. 28, no. 1, pp. 33–44, 2020, doi: 10.1093/jigpal/jzz069.
[74] *Romanian and Swiss TSP Instances*. Accessed: Mar. 1, 2020. [Online]. Available: http://cadredidactice.ub.ro/ceraselacrisan/cercetare/
[75] *DIMACS Implementation Challenge. TSP*. Accessed: Apr. 28, 2020. [Online]. Available: http://dimacs.rutgers.edu/archive/Challenges/TSP/download.html
[76] L. Sengupta and P. Franti, "Predicting the difficulty of TSP instances using MST," in *Proc. IEEE 17th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2019, pp. 847–852, doi: 10.1109/INDIN41052.2019.8972232.
[77] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*. Amsterdam, The Netherlands: Elsevier, 2004.
[78] S. Wright, "The roles of mutation, inbreeding, crossbreeding and selection in evolution," *Proc. 6th Int Congr. Genet.*, vol. 1, 1932, pp. 356–366.
[79] G. Ochoa and N. Veerapen, "Mapping the global structure of TSP fitness landscapes," *J. Heuristics*, vol. 24, no. 3, pp. 265–294, Jun. 2018, doi: 10.1007/s10732-017-9334-0.
[80] G.-G. Wang, "Moth search algorithm: A bio-inspired Metaheuristic algorithm for global optimization problems," *Memetic Comput.*, vol. 10, no. 2, pp. 151–164, Jun. 2018, doi: 10.1007/s12293-016-0212-3.

GLORIA CERASELA CRIŞAN received the degree in informatics from the University of Bucharest, Romania, in 1986, and the Ph.D. degree in informatics from the Alexandru Ioan Cuza University of Iaşi, Romania, in 2008.

Since 2016, she has been an Associate Professor with the Department of Mathematics and Informatics, Faculty of Sciences, Vasile Alecsandri University of Bacău, Romania. Her research interests include combinatorial optimization problems, metaheuristics, transportation and logistics problems, and GIS.

ELENA NECHITA received the degree in informatics and the Ph.D. degree in informatics from the Alexandru Ioan Cuza University of Iaşi, Romania, in 1987 and 2000, respectively.

Since 2015, she has been a Professor with the Department of Mathematics and Informatics, Faculty of Sciences, Vasile Alecsandri University of Bacău, Romania. Her research interests include artificial intelligence, probability and statistics, computers, and education.

DANA SIMIAN received the B.S. degrees in mechanical engineering and mathematics in 1994 and 1984, respectively, and the Ph.D. degree in mathematics from the Babeş-Bolyai University of Cluj-Napoca, Romania, in 2001.

She was a Visiting Professor with the University of Cincinnati, Ohio, USA, in 2018, and the University of Applied Sciences, Wurzburg-Schweinfurt, Germany, from 2017 to 2019. She is currently a Professor with the Department of Mathematics and Informatics, Faculty of Sciences, Lucian Blaga University of Sibiu, Romania. She is also the Director of the Research Center on Informatics and Information Technology, Faculty of Science. Her research interests include machine learning, modeling and optimization, algorithms and data structures, numerical calculus, and computational geometry.