axioms

MDPI

Article

A Multi-Phase Method for Euclidean Traveling Salesman Problems

Víctor Hugo Pacheco-Valencia [1], Nodari Vakhania [1,*], Frank Ángel Hernández-Mira [2] and José Alberto Hernández-Aguilar [3]

[1] Centro de Investigación en Ciencias UAEMor, Universidad Autónoma del Estado de Morelos, Av. Universidad 1001, Col.Chamilpa, Cuernavaca 62209, Mexico
[2] Facultad de Matemáticas UAGro, Universidad Autónoma de Guerrero, Carlos E. Adame No. 54, Col.Garita, Acapulco 39650, Mexico
[3] Facultad de Contaduría, Administración e Informática UAEMor, Universidad Autónoma del Estado de Morelos, Av. Universidad 1001, Col.Chamilpa, Cuernavaca 62209, Mexico
* Correspondence: nodari@uaem.mx

Abstract: The Traveling Salesman Problem (TSP) aims to find the shortest tour for a salesman who starts and ends in the same city and visits the remaining $n-1$ cities exactly once. There are a number of common generalizations of the problem including the Multiple Traveling Salesman Problem (MTSP), where instead of one salesman, there are k salesmen and the same amount of individual tours are to be constructed. We consider the Euclidean version of the problem where the distances between the cities are calculated in two-dimensional Euclidean space. Both general the TSP and its Euclidean version are strongly NP-hard. Hence, approximation algorithms with a good practical behavior are of primary interest. We describe a general method for the solution of the Euclidean versions of the TSP (including MTSP) that yields approximation algorithms with a favorable practical behavior for large real-life instances. Our method creates special types of convex hulls, which serve as a basis for the constructions of our initial and intermediate partial solutions. Here, we overview three algorithms; one of them is for the bounded version of the MTSP. The proposed novel algorithm for the Euclidean TSP provides close-to-optimal solutions for some real-life instances.

Keywords: traveling salesman problem; heuristic algorithm; time complexity; convex hull; two-dimensional Euclidean space; optimization

MSC: 11A05; 52B55

Citation: Pacheco-Valencia, V.H.; Vakhania, N.; Hernández-Mira, F.Á.; Hernández-Aguilar, J.A. A Multi-Phase Method for Euclidean Traveling Salesman Problems. *Axioms* **2022**, *11*, 439. https://doi.org/10.3390/axioms11090439

Academic Editors: Luigi Brugnano and Simeon Reich

Received: 18 July 2022
Accepted: 26 August 2022
Published: 30 August 2022

1. Introduction

The Traveling Salesman Problem (TSP) is an NP-hard problem, as proven by Karp in 1972 [1]. It aims to construct a tour for a salesman who must visit n clients or cities, each of them exactly once, except that the first visited city is repeatedly visited at the end of the tour. The objective is to minimize the overall incurred distance. We shall rely on the following graph-theoretical formulation of the problem. We are given an undirected weighted complete graph $G = (V, E)$, where the vertices from set V are identified as the clients or cities and the edges from set E represent connections between these cities, so that their weights represent the corresponding distances; i.e., for every $(i, j) \in E$, the weight $w(i, j)$ is the distance between cities i and j. We consider the Euclidean setting of the problem, where the distances between the cities are calculated in a two-dimensional plane, the cities being represented as points in that plane. Therefore, we treat the vertices from set V as points in the two-dimensional plane and edges as vectors in that plane. In this way, we deal with a kind of projection of graph G in the two-dimensional plane. Note that the Euclidean TSP is symmetric, i.e., for any edge (i, j), $w(i, j) = w(j, i)$.

A permutation of the n points defines a feasible tour $T = (i_1, \ldots, i_n, i_1)$ in the two-dimensional plane. The cost of tour T, $\mathrm{C}(T)$ is

$$\mathrm{C}(T) = \mathrm{w}(i_1, i_2) + \mathrm{w}(i_2, i_3) + \cdots + \mathrm{w}(i_{n-1}, i_n) + \mathrm{w}(i_n, i_1) \quad (1)$$

The objective is to find a feasible solution T^* whose cost $\mathrm{C}(T^*)$ is the minimum, i.e., $\mathrm{C}(T^*) = \min_T \mathrm{C}(T)$.

The Multiple Traveling Salesman Problem (MTSP) is a generalization of the TSP, which deals with $k \geq 1$ salesmen and a specific city d called the *depot*. Correspondingly, k different tours are to be constructed such that each tour starts and ends at the depot. The *bounded* version of the MTSP (BMTSP) imposes a restriction on the number of cities each tour has to include where a feasible solution consists of k tours $\{T^1, T^2, \cdots, T^k\}$, where

$$T^j = (d, i_1^j, i_2^j, \cdots, i_{m_j}^j, d) \quad (2)$$

An integer m_j is the number of cities that tour T^j has to include such that $m_1 + m_2 + \cdots + m_k = n$.

The cost of a feasible solution $\{T^1, T^2, \cdots, T^k\}$ is

$$\mathrm{C}(\{T^1, T^2, \cdots, T^k\}) = \mathrm{C}(T^1) + \mathrm{C}(T^2) + \cdots + \mathrm{C}(T^k) \quad (3)$$

The aim in studying the BMTSP with several salesmen and with upper and lower bounds on the number of clients for each salesman is not actually to attain a better total cost compared to the case when a single salesman is used. The motivation here is to reduce the time needed to serve all the clients taking into account the limited capacity of each salesman (vehicle). Hence, an upper bound on the number of clients for every salesman is imposed, where a lower bound somehow "balances" the minimal load for each salesman. These considerations were first reported by Roerty [2] and later detailed by Necula [3].

In this study, we focus on the Euclidean version of the TSP, which remains strongly NP-hard [4]. Nevertheless, it is more "transparent" than the general version, in the sense that simple geometric considerations can be used in a solution process of the Euclidean version (see, e.g., [5]). Here, we describe our method, based on simple geometric observations, and three simple, fast, and easily implementable direct combinatorial algorithms yielded by the method.

Our method uses special types of convex hulls, which serve as a basis for the constructions of our initial and intermediate partial solutions. A basic type of convex hull that we employ is a *girding polygon* P [6], the minimal convex polygon that includes all points from set V, a convex hull for set V.

The method starts with a (partial) tour defined by the girding polygon P. If this tour is complete, i.e., contains all points from set V, then it is also an optimal tour. Otherwise, yet uncovered cities are iteratively added to the current partial solution until a feasible solution is constructed. This feasible solution is further improved using local search algorithms. This framework for the TSP is extended for the BMTSP using a preliminary stage that partitions the points from set V into k feasible subsets. The second proposed algorithm for the Euclidean TSP constructs convex hulls for the subsets of vertices from set V, convex polygons. These polygons may intersect each other. Every created polygon defines a partial tour. Iteratively, a partial tour defined by the polygon of a current iteration is unified with the partial tour of the previous iteration, resulting in the partial tour of the current iteration. In this way, the polygons and the tours are iteratively "inflated".

We found a few algorithms that also use convex hulls in one way or another. For example, Macgregor and Ormerod in 1996 [7] tested the hypothesis that the difficulty of solving a TSP for humans is a function of the number of interior points of the corresponding girding polygon. They designed two experiments with instances of the TSP. The first experiment used instances with ten points, and the second one used instances with twenty points in total. These instances were solved by a group of people manually. These solutions were

compared with the solutions obtained by three common heuristics, the nearest neighbor algorithm, the largest interior angle algorithm (Norback and Love in 1977 [8]), and a variation of a convex hull heuristic described by Golden et al. in 1980 [9] using the least expensive insertion criterion. Notably, the manually obtained solutions turned out to be better than the ones created by the heuristics. Sengupta and Fränti [10] continued this line of research, measuring the dependence of the complexity of the solutions obtained by the humans on the number of interior points of the polygon.

We also found several algorithms in the literature for the construction of a girding polygon. The algorithm from Andrew [11] reorders the set of vertices according to their x and y coordinates, which are used in the algorithm for the construction of the girding polygon. In [6,8,12–14], The girding polygon is constructed based angles of the girding polygon based on the calculation of specially determined angles. In [15–17], the set of vertices were iteratively partitioned into two subsets separated with a line, formed by the two specially determined points. The furthest from the constructed line vertex of one or the other subset was identified as the next point that was be included in the formed girding polygon. In [18–20], the construction algorithms that use "divide and conquer" techniques was presented, where a merge algorithm for two non-intersecting convex hulls was recursively applied.

Our algorithm for the construction of the girding polygon, can briefly be described as follows. It initially identifies four specially determined auxiliary points on the plane, one of which forms the partial polygon of the initial iteration 1. The girding polygon is formed in ν consecutive iterations, where $\nu \leq n$ is the number of vertices of polygon P. At each iteration $h > 1$, a subset of vertices from set V is associated with each of the auxiliary points. This division of the set V at iteration h is performed according to the coordinates of the four auxiliary points and the coordinates of the point added to the partial polygon at iteration $h-1$. At iteration h, a new point from one of the four subsets is added to the partial polygon of iteration $h-1$. This new point is chosen based on some geometric calculations. A brief description of this algorithm was originally given in [6], where MTSP is solved in two phases. In the first phase, after the construction of girding polygon P, set V is partitioned into k subsets using specially defined k auxiliary edges (d, x), here $x \in P$ and k is the number of salesmen. An algorithm is proposed for the construction of a feasible tour for TSP, that is applied for each of the subsets at the second phase.

Next, we give a brief overview of some related work and applications. Jünger, Reinelt, and Rinaldi [21] published an overview in which they described applications of the TSP in real-life problems and exposed different heuristics and approximation algorithms. Possible applications occur in the drilling of printed circuit boards, in the analysis of the structure of crystals (Bland and Shallcross [22]), during material handling in a warehouse (Ratliff and Rosenthal [23]), in overhauling gas turbine engines (Plante, Lowe, and Chandrasekaran [24]), in the order-picking problem in warehouses (Ratliff and Rosenthal [23]), in constructing computer boards to optimize the connections between different board components, and in scheduling problems where job process times are sequence-dependent (Lenstra and Rinnooy Kan [25]).

Bektas [26] described different existing variants of the MTSP and their relationship with other problems; he also mentioned some other practical applications. Cheikhrouhou and Khoufi [27] published a comprehensive survey in which they included the approaches applied to unmanned aerial vehicles and/or drones; they also classified some practical applications for the MTSP: monitoring and surveillance, network connectivity, search and rescue, precision agriculture, and disaster management. Other applications arise in scheduling a printing press (Gorenstein [28]), in mission planning (Brumitt and Stentz [29]), in interview scheduling (Gilbert and Hofstra [30]), in crew scheduling, in school bus routing [31], in cash distribution in banks, and in recollecting cash from telephone booths and their repair (see Svestka and Huckfeldt [32] and Lenstra and Rinnooy Kan [25]). Other applications occur in the planning of autonomous robots (Yu et al. [33]) and in planning of unmanned aerial vehicle applications (Ryan et al. [34]). An application of the MTSP arises

in the optimal design of global navigation satellite systems (Saleh and Chelouah [35]). As for the bounded version BMTSP, there exist meta-heuristic algorithms for its solution (see, for example, [3,36–40]).

We conclude this section with a brief description of how this paper is organized. In Section 2, we describe our general method and three particular algorithms that we constructed so far using our method. In Section 2.1, we give a detailed description of our algorithm that generates the girding polygon P. In Section 2.2, the insertion algorithm from [5] is described. We describe our basic algorithm for the BMTSP in Section 2.3, and we present another novel inflammation algorithm in Section 2.4. Section 3 presents the results of our experimental study. The last Section 4 contains some concluding remarks.

Part of the material from this work was presented at the 1st International Online Conference on Algorithms (IOCA 2021) (see [41,42]).

2. Methods

2.1. Construction of Girding Polygon P

Our girding polygon P can be seen as a convex geometric figure in 2-dimensional Euclidean space with the edges from the projection of graph G enclosing all vertices of set V. Polygon P can also be seen as a sub-graph of graph G. Note that polygon P immediately defines an optimal tour T_0 for its ν vertices. Our construction algorithm iteratively extends this tour until it creates a complete feasible tour.

Figure 1 represents a small problem instance "*tsp10*" with 10 vertices, which we use for the illustration purposes throughout this paper.

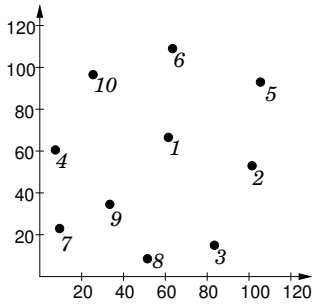


Figure 1. Example instance "*tsp10*" with 10 vertices.

Our algorithm for the construction of the girding polygon P works in a number of iterations, P_h being the *partial polygon* (polygonal line) of iteration h. Iteratively, at iteration h, one vertex, denoted by p_h, is added to the partial polygon P_{h-1} of iteration $h-1$, i.e., $P_h = P_{h-1} \cup \{p_h\}$ (abusing again the notation, we also use P_h for the set of vertices in polygon P_h).

Let $i \in P_{h-1}$ and $j \in V \setminus P_{h-1}$, and let $(i,j) \in E$ be the corresponding vector (edge) in 2-dimensional Euclidean space. The algorithm for the construction of polygon P uses function $\theta(i,j)$ (Equation (4)), which returns the angle between axes $y = y_i$ and vector (i,j); see Figure 2.

$$\theta(i,j) = \begin{cases} \arccos \dfrac{x_j - x_i}{\mathrm{w}(i,j)} & if \ \arcsin \dfrac{y_j - y_i}{\mathrm{w}(i,j)} \geq 0 \\ -\arccos \dfrac{x_j - x_i}{\mathrm{w}(i,j)} & if \ \arcsin \dfrac{y_j - y_i}{\mathrm{w}(i,j)} < 0 \end{cases} \tag{4}$$

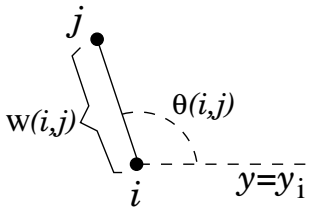


Figure 2. Angle $\theta(i,j)$ formed by the edge (i,j) and the straight line $y = y_i$.

Now, we turn to the description of the main algorithm that constructs polygon P. It works in four stages, each stage being associated with one of the specially determined auxiliary points in 2-dimensional Euclidean space. We denote these auxiliary points by v^1, v^2, v^3, and v^4 and will refer to them as the "*uppermost*", the "*leftmost*", the "*lowermost*", and the "*rightmost*" points, respectively [6]; see Figure 3. More formally,

$$v^1 = j_1 | x_{j_1} \textit{ is maximum and } j_1 \in \{i_1 | y_{i_1} \textit{ is maximum and } i_1 \in V\}$$
$$v^2 = j_2 | y_{j_2} \textit{ is maximum and } j_2 \in \{i_2 | x_{i_2} \textit{ is minimal and } i_2 \in V\}$$
$$v^3 = j_3 | x_{j_3} \textit{ is minimal and } j_3 \in \{i_3 | y_{i_3} \textit{ is minimal and } i_3 \in V\}$$
$$v^4 = j_4 | y_{j_4} \textit{ is minimal and } j_4 \in \{i_4 | x_{i_4} \textit{ is maximum and } i_4 \in V\}$$

Below, we illustrate how the four auxiliary points are determined for our problem instance "tsp10".

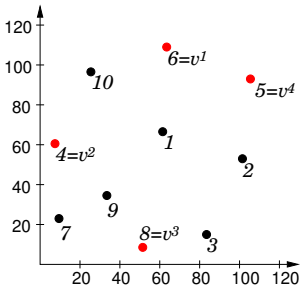


Figure 3. The four auxiliary points $v^1 = 6$, $v^2 = 4$, $v^3 = 8$, and $v^4 = 5$ determined for example instance "tsp10".

Roughly, some points located in between points v^1 and v^2 are added at stage 1; some points located in between points v^2 and v^3 are added at stage 2; some points located in between points v^3 and v^4 are added at stage 3; some points located in between points v^4 and v^1 are added at stage 4. Correspondingly, the whole polygon area is divided into four subareas V^l, $l = 1, \ldots, 4$, defined as follows:

$$V^1 = \{k \mid x_k < x_{p_{h-1}} \wedge y_k \geq y_{v^2} \quad ; k \in V\}$$
$$V^2 = \{k \mid x_k \leq x_{v^3} \wedge y_k < y_{p_{h-1}} \quad ; k \in V\}$$
$$V^3 = \{k \mid x_k > x_{p_{h-1}} \wedge y_k \leq y_{v^4} \quad ; k \in V\}$$
$$V^4 = \{k \mid x_k \geq x_{v^1} \wedge y_k > y_{p_{h-1}} \quad ; k \in V\}$$

The algorithm works in a number of iterations. Initially, $P_0 = (v^1)$, i.e., the initial partial polygon (polygonal line) consists of the uppermost vertex v^1. At stage l, at iteration $h > 0$, $p_h \in V^l$ is the vertex, added to the set P_{h-1}, resulting in an augmented polygonal line (set) P_h of iteration h. In particular, vertex $p_h \in V^l$ is such that

$$\theta(p_{h-1}, p_h) = \min_{j \in V^l} \theta(p_{h-1}, j).$$

Omitting some minor details, below, we give a formal description of the algorithm and its flowchart (Figure 4):

We illustrate the flowchart of Algorithm 1 (*GIRDING_POLYGON*) for the problem instance "tsp10" in Table 1. Column "h" is the current iteration, "l" is the current stage, and "condition" reflects the "while" condition.

In Figure 5, we illustrate the flowchart of Algorithm 1 (*GIRDING_POLYGON*) in graphical form for the same problem instance.

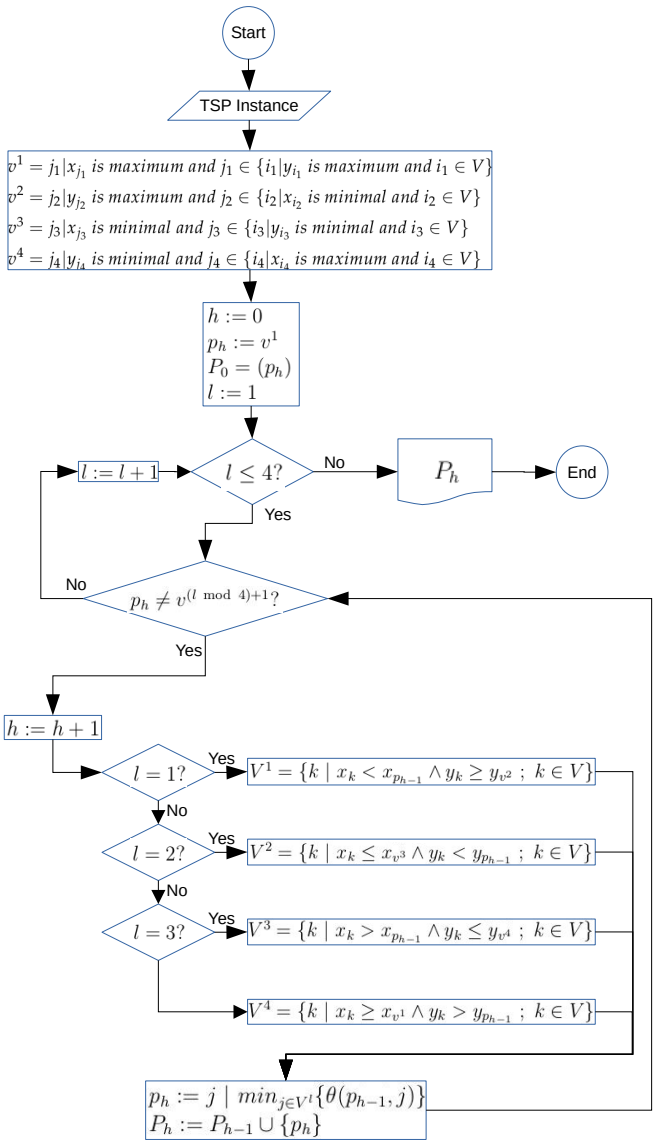


Figure 4. Girding polygon algorithm flowchart.

Algorithm 1: *GIRDING_POLYGON* (V).

$h := 0$
$p_h := v^1$
$P_0 := (p_h)$
for $l := 1$ **to** 4 **do**
  while $p_h \neq v^{(l \bmod 4)+1}$ **do**
    $h := h + 1$
    if $l = 1$ **then** $V^1 = \{k \mid x_k < x_{p_{h-1}} \wedge y_k \geq y_{v^2}\ ;\ k \in V\}$
    if $l = 2$ **then** $V^2 = \{k \mid x_k \leq x_{v^3} \wedge y_k < y_{p_{h-1}}\ ;\ k \in V\}$
    if $l = 3$ **then** $V^3 = \{k \mid x_k > x_{p_{h-1}} \wedge y_k \leq y_{v^4}\ ;\ k \in V\}$
    if $l = 4$ **then** $V^4 = \{k \mid x_k \geq x_{v^1} \wedge y_k > y_{p_{h-1}}\ ;\ k \in V\}$
    $p_h := j \mid min_{j \in V^l}\{\theta(p_{h-1}, j)\}$
    $P_h := P_{h-1} \cup \{p_h\}$
return P_h

Table 1. Iterations performed by the Algorithm 1 (*GIRDING_POLYGON*) for the construction of the girding polygon P for the example instance "tsp10".

h	l	p_{h-1}	$v^{(l \bmod 4)+1}$	*Condition*	V^l	p_h	$P_h := P_{h-1} \cup \{p_h\}$
0						6	(6)
1	1	6	4	$6 \neq 4$?: True	$\{1, 10, 4\}$	10	(6, 10)
2	1	10	4	$10 \neq 4$?: True	$\{4\}$	4	(6, 10, 4)
3	1	4	4	$4 \neq 4$?: False			
3	2	4	8	$4 \neq 8$?: True	$\{7, 8, 9\}$	7	(6, 10, 4, 7)
4	2	7	8	$7 \neq 8$?: True	$\{8\}$	8	(6, 10, 4, 7, 8)
5	2	8	8	$8 \neq 8$?: False			
5	3	8	5	$8 \neq 5$?: True	$\{1, 2, 3, 5\}$	3	(6, 10, 4, 7, 8, 3)
6	3	3	5	$3 \neq 5$?: True	$\{2, 5\}$	2	(6, 10, 4, 7, 8, 3, 2)
7	3	2	5	$2 \neq 5$?: True	$\{5\}$	5	(6, 10, 4, 7, 8, 3, 2, 5)
8	3	5	5	$5 \neq 5$?: False			
8	4	5	6	$5 \neq 6$?: True	$\{6\}$	5	(6, 10, 4, 7, 8, 3, 2, 5, 6)

2.2. The Basic 3-Phase Insertion Algorithm for TSP

In this section, we overview a 3-phase algorithm from [5]. It starts with phase 1 by constructing girding polygon P as described in Section 2.1. Recall that this polygon defines an initial (partial) tour T_0. If that tour is complete, i.e., contains all vertices from set V, then nothing remains to be done; the algorithm halts with this optimal tour. Otherwise, this tour is iteratively extended at phase 2 by Algorithm 2 (see Figure 6).

Let T_h be the partial tour constructed by iteration h. The algorithm iteratively, at iteration $h > 0$, inserts a new, yet non-covered vertex $l^h \in V \setminus T_{h-1}$ into the current tour T_{h-1}, one that yields the minimal increase in the cost of the formed extended tour $T_h = T_{h-1} \cup \{l^h\}$ of iteration h (abusing slightly the notation, here and later, we may use T_h for the set of vertices from tour T_h). The insertion algorithm stops when it constructs a complete feasible tour including all vertices from set V. In the figure below, we illustrate how the algorithm works for our problem instance.

Figure 7 illustrates the three iterations of the insertion algorithm for our problem instance "tsp10" with 10 vertices. Here, $T_0 = (6, 10, 4, 7, 8, 3, 2, 5, 6)$. At iteration 1, the costs C_9^1 and C_1^1 are calculated, and the vertex 9 is inserted between the vertices 7 and 8 since $C_9^1 < C_1^1$; see Figure 7b. At iteration 2, C_1^2 is (re)calculated and the vertex 1 is inserted between vertices 5 and 6; see Figure 7c. Hence, a feasible solution $(6, 10, 4, 7, 9, 8, 3, 2, 5, 1, 6)$ is delivered. The pseudo-code of the insertion algorithm and its flowchart are given below.

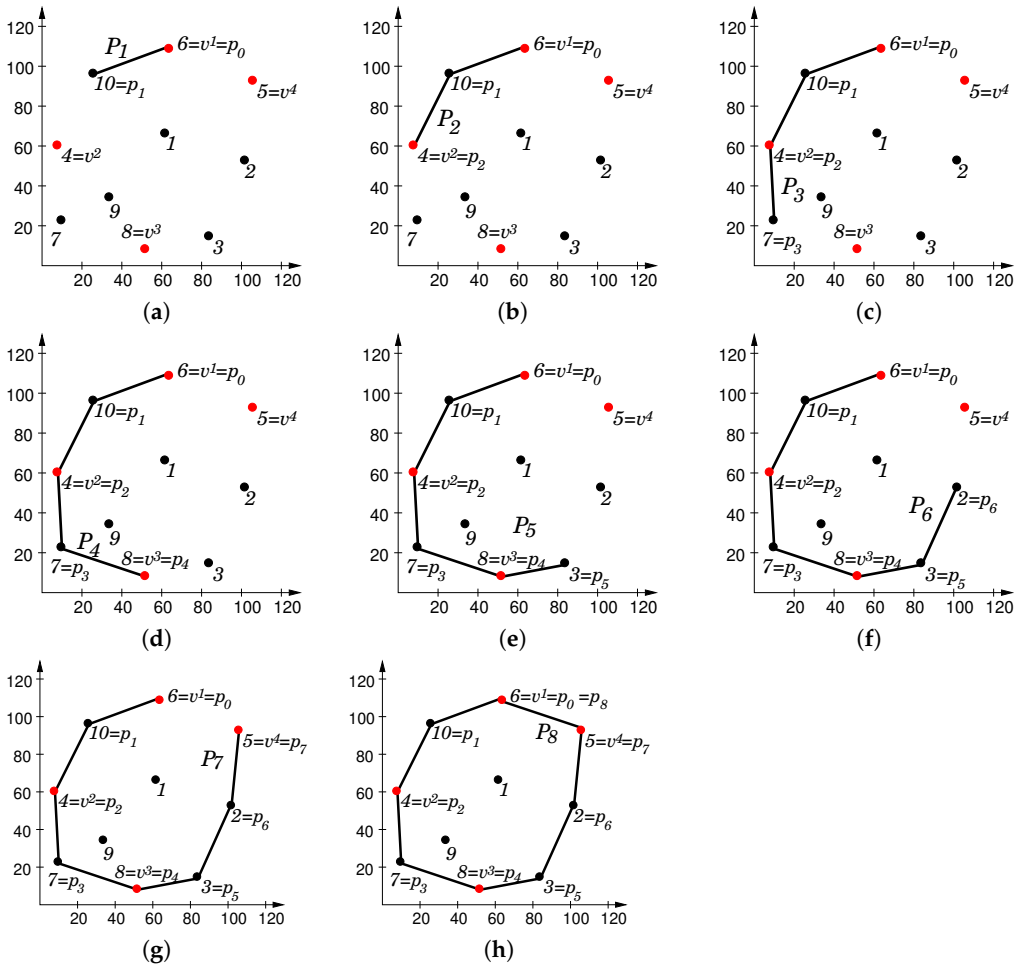


Figure 5. Polygonal lines constructed by algorithm (*GIRDING_POLYGON*) for instance "tsp10". (**a**) $P_1 = (6, 10)$; (**b**) $P_2 = (6, 10, 4)$; (**c**) $P_3 = (6, 10, 4, 7)$; (**d**) $P_4 = (6, 10, 4, 7, 8)$; (**e**) $P_5 = (6, 10, 4, 7, 8, 3)$; (**f**) $P_6 = (6, 10, 4, 7, 8, 3, 2)$; (**g**) $P_7 = (6, 10, 4, 7, 8, 3, 2, 5)$; (**h**) $P_8 = (6, 10, 4, 7, 8, 3, 2, 5, 6)$.

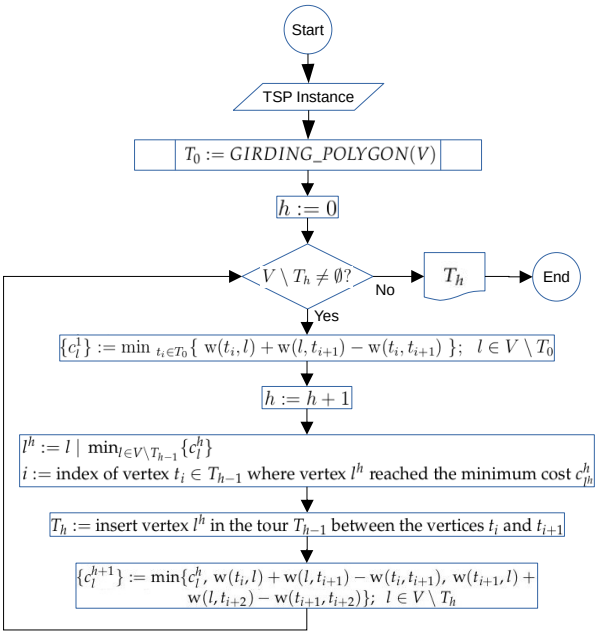


Figure 6. Insertion algorithm flowchart.

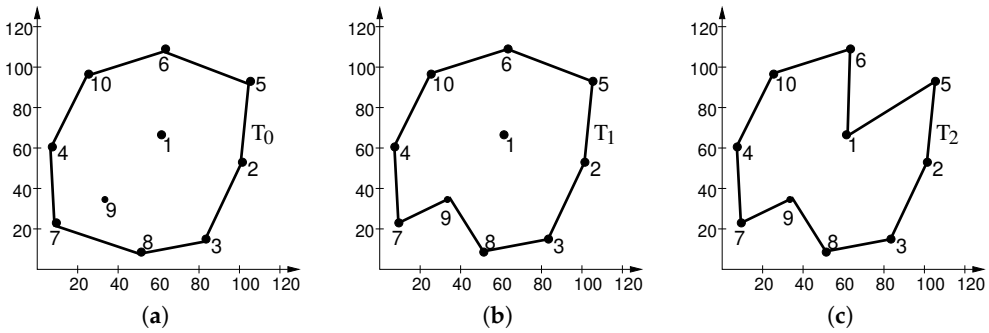


Figure 7. Iterations made by the insertion algorithm to solve the example instance "tsp10". (**a**) The tour $T_0 = (6, 10, 4, 7, 8, 3, 2, 5, 6)$ is delivered by $(GIRDING\_POLYGON(V))$. (**b**) Vertex 9 is inserted between vertices 7 and 8. (**c**) Vertex 1 is inserted between vertices 5 and 6. The feasible tour constructed is $(6, 10, 4, 7, 9, 8, 3, 2, 5, 1, 6)$.

Algorithm 2: $INSERTION(V)$.

$h := 0$
$T_0 := GIRDING\_POLYGON(V)$
if $V \setminus T_0 \neq \emptyset$ **then**
  $\{c_l^1\} := \min_{t_i \in T_0} \{ \mathrm{w}(t_i, l) + \mathrm{w}(l, t_{i+1}) - \mathrm{w}(t_i, t_{i+1}) \};\ l \in V \setminus T_0$
  while $V \setminus T_h \neq \emptyset$ **do**
    $h := h + 1$
    $l^h := l \mid \min_{l \in V \setminus T_{h-1}} \{c_l^h\}$
    $i :=$ index of vertex $t_i \in T_{h-1}$, where vertex l^h reached the minimum cost $c_{l^h}^h$
    $T_h :=$ insert vertex l^h in the tour T_{h-1} between the vertices t_i and t_{i+1}
    $\{c_l^{h+1}\} := \min\{c_l^h,\ \mathrm{w}(t_i, l) + \mathrm{w}(l, t_{i+1}) - \mathrm{w}(t_i, t_{i+1}),\ \mathrm{w}(t_{i+1}, l) + \mathrm{w}(l, t_{i+2}) - \mathrm{w}(t_{i+1}, t_{i+2})\};\ l \in V \setminus T_h$
return T_h

At phase 3, the algorithm iteratively improves the solution delivered by phase 2 using a specially constructed local improvement algorithm.

As is easy to verify, phase 1 runs in time $O(n\nu)$, where ν is the number of vertices of the polygon P. Phase 2 needs at most $n - 3$ iterations, where the selection cost at each iteration is $O(n)$. The improvement phase has an amortized time complexity of $O(n^2)$.

2.3. Partition, Construction, and Improvement Algorithm for the BMTSP

This section contains a brief description of our algorithm for the bounded version of the MTSP, BMTSP [42], this algorithm, roughly, extends the insertion algorithm for the TSP from the previous section with an additional partition phase. Our partition method adopts and modifies one from [6].

At the partition phase, we partition set $V \setminus \{d\}$ in k subsets $V_1, V_2, \cdots, V_k$ such that each subset respects the problem restriction , i.e., the number of vertices in each of them is within the range $[m_{\min}, m_{\max}]$).

Our partition algorithm first defines an auxiliary point c with the coordinate x_c (y_c, respectively) being the average of the x-coordinates (y-coordinates, respectively) of the vertices from set V:

$$x_c = \frac{\sum_{i \in V \setminus \{d\}} x_i}{|V \setminus \{d\}|},$$

and

$$y_c = \frac{\sum_{i \in V \setminus \{d\}} y_i}{|V \setminus \{d\}|}.$$

Then, k additional auxiliary points $i'_1, \cdots, i'_k$ are defined. The first of them, $i'_1 \in V \setminus \{d\}$ is the farthest point from the depot. The remaining $k-1$ auxiliary points $i'_2, \cdots, i'_k$ are determined in such a way that the angle between two adjacent vectors (c, i'_j) and (c, i'_{j+1}) is the same and equals $\frac{2\pi}{k}$ and that all vectors (c, i'_j) have the same length $\mathrm{w}(c, i'_1)$ (see Figure 8).

The partition of the set $V \setminus \{d\}$ is now carried out in two stages. In stage 1, each subset V_j, $j = 1, \ldots, k$, contains a single vertex $l_j \in V \setminus \{\{V_j\} \cup \{d\}\}$ such that the distance between l_j and the auxiliary point i'_j is the minimal possible among all yet non-selected vertices of set $V \setminus \{d\}$.

Once the initial sets V_j, $j = 1, \ldots, k$ are so formed, repeatedly, for each subset V_j, a new vertex v with the minimum distance between that vertex and a vertex from the current set V_j are added to that subset, i.e., $V_j := V_j \cup \{v\}$. This operation is repeated until the current set V_i contains $m_{\min}$ elements (see Figure 9).

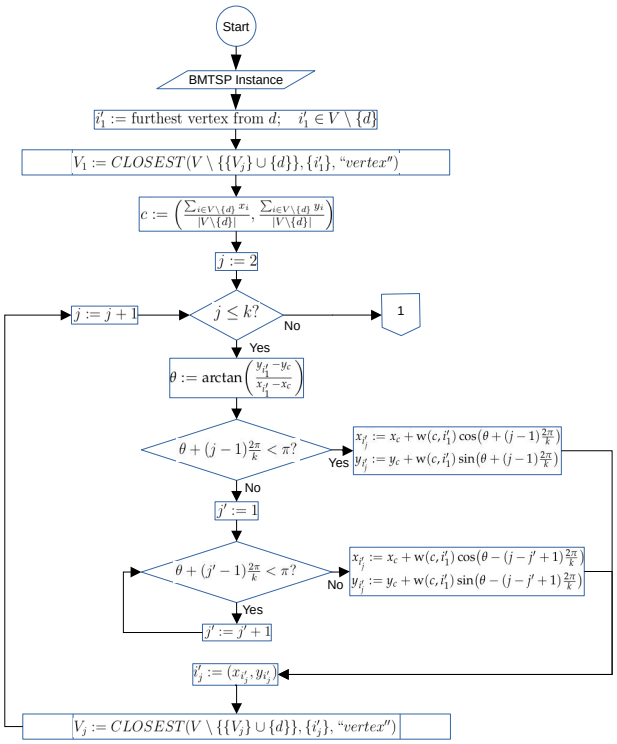


Figure 8. A basic flowchart of the partition algorithm to obtain the auxiliary points $i'_1, \cdots, i'_k$ and the first vertex for each subset V_j, $j = 1, \cdots, k$.

In stage 2, the subsets of stage 2 are augmented. For every yet non-selected vertex v, a “closest” subset V_j is looked for, i.e., a subset such that the distance between vertex v and a point from set V_j is the minimal possible among all subsets from the current partition. This operation is repeated until there remains a non-selected vertex (see Figure 10).

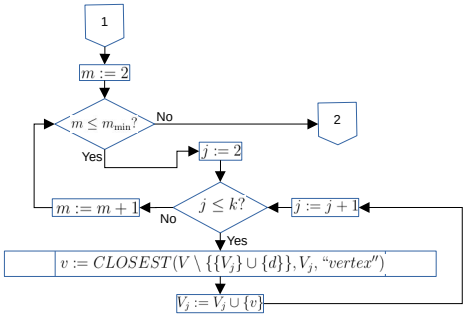


Figure 9. A partial flowchart for the creation of the subsets V_j, $j = 1, \cdots, k$ with $m_{\min}$ vertices.

Once the above partition process is complete, a feasible tour T_j for each of the subsets $V_j \cup \{d\}$ is constructed by merely invoking the insertion algorithm from Section 2.2 (see Figure 11). This solution is iteratively improved by a local interchange algorithm, which iteratively selects a vertex from tour T^j and alters its position within that tour or this vertex is relocated to another tour $T^{j'}$. The exchange is accomplished if it yields the decrease of the total cost; among all such exchanges, one which yields the maximum decrease is accomplished so that the resultant solution remains feasible.

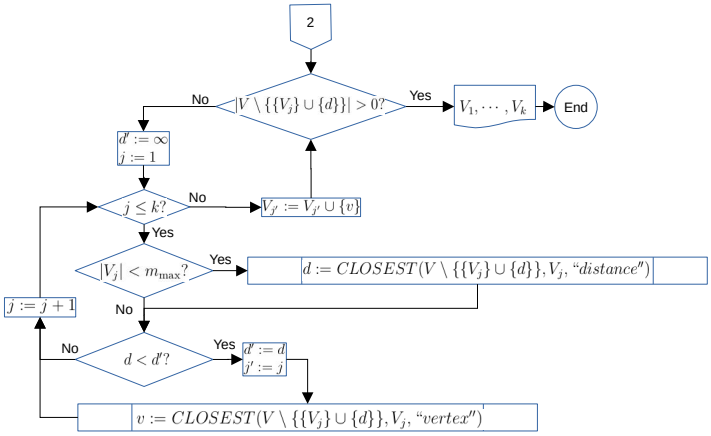


Figure 10. The last partial flowchart of the partition algorithm to obtain the subsets V_j, $j = 1, \cdots, k$.

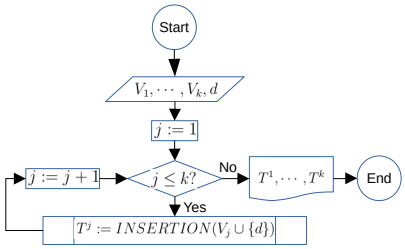


Figure 11. The flowchart of the construction algorithm to obtain the k tours T_j, $j = 1, \cdots, k$.

Figure 12 illustrates the algorithm applied to our sample problem instance with 10 vertices. Here, we let $k = 3$ salesmen, $m_{\min} = 2$, and $m_{\max} = 4$. In Figure 12a, the auxiliary point i'_1 is 7. Then, the coordinates of auxiliary points i'_2 and i'_3 are calculated. Figure 12b represents the created partition of set $V \setminus \{d\}$ with $V_1 = \{3, 7, 8, 9\}$, $V_2 = \{2, 5\}$ and $V_3 = \{4, 6, 10\}$. In Figure 12c, tours $T^1 = (d, 9, 7, 8, 3, d)$, $T^2 = (d, 2, 5, d)$, and $T^3 = (d, 6, 10, 4)$ are represented.

As is easy to verify, phase 1 runs in time $O(nkm^2_{\max})$; phase 2 has the same time complexity as the improvement phase of the previous algorithm; phase 3 runs in time $O(k^2 m^2_{\max})$; see [42].

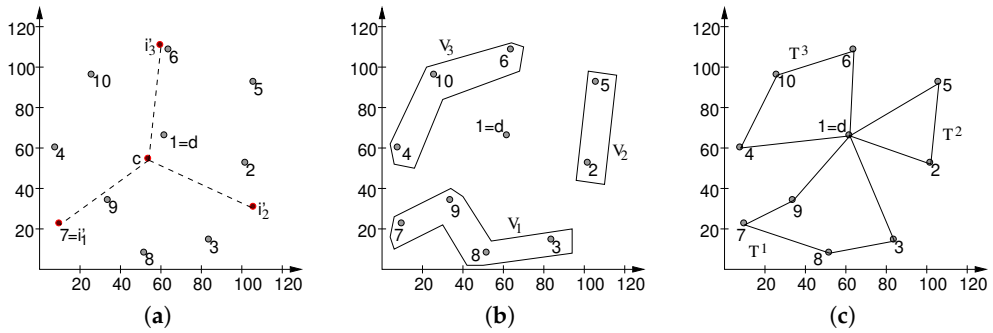


Figure 12. Phases to construct a solution for an instance of the bounded MTSP with $k = 3$, $m_{\min} = 2$ and $m_{\max} = 4$.

2.4. The Inflammation Algorithm

In this section, we describe our novel algorithm for the basic TSP that we call an *inflammation* algorithm. The name comes from the general approach used in the algorithm that iteratively constructs specially defined polygons delineating larger and larger areas in the plane with more and more points from set V. Each of these polygons defines a partial tour including a progressively increasing number of vertices.

Unlike the insertion algorithm, the first polygon P_0 that the inflammation algorithm constructs is the "smallest" one that is extended to "larger" polygons at consecutive iterations (in particular, it is not a girding polygon). The polygon constructed at iteration h, P_h, is an *inner convex hull*, i.e., it does not contain vertices from set V in its interval area, except that it may contain vertices in its internal area that belong to the earlier-constructed polygons $P_0, \ldots, P_{h-1}$ (if polygon P_h contains a vertex from set V in its internal area, then this vertex belongs to the polygon of an earlier iteration). Note that each polygon P_h defines an optimal partial tour for the set of vertices that it contains, similarly to how the girding polygon P from Section 2.1 defines an optimal tour T_0. As we know, if $V \setminus T_0 = \emptyset$, then T_0 from Section 2.1 defines an optimal tour. Hence, in the inflammation algorithm, we assume that this condition is not satisfied.

The inflammation algorithm iteratively, at every iteration h, constructs the polygon P_h containing (some) vertices from set $V \setminus T_{h-1}$ (see Algorithm 3), where we denote again by T_h the partial tour constructed by iteration h. Initially, we let $T_0 = \emptyset$, and T_1 is formed by the partial tour of polygon P_1. At iteration $h \geq 2$, Algorithm 4 ($JOINP$) constructs partial tour T_h by joining the vertices of polygon P_h and the partial tour T_{h-1} as described later in Section 2.4.2.

This completes an aggregated description of the inflammation algorithm. Abusing again the notation, we may use P_h for the set of vertices of polygon P_h.

We shall refer to a specially defined point $c = (x_c, y_c)$ in the two-dimensional plane as a *centroid*.

In the following two subsections, we describe Algorithm 5, ($INNER\_CONVEX\_HULL$), which constructs inner convex hull P_h, and Algorithm 4 ($JOINP$), which creates partial tour T_h from the polygon P_h and the previous tour T_{h-1}, at every iteration h.

Algorithm 3: $INFLAMMATION(V)$.

$h := 0$
$T_0 := \emptyset$
while $|V \setminus T_h| > 2$ **do**
  $h := h + 1$
  $c_h := \left(\frac{\sum_{i \in V \setminus T_{h-1}} x_i}{|V \setminus T_{h-1}|}, \frac{\sum_{i \in V \setminus T_{h-1}} y_i}{|V \setminus T_{h-1}|} \right)$
  $P_h := INNER\_CONVEX\_HULL(V \setminus T_{h-1}, c_h)$
  $T_h := JOINP(T_{h-1}, P_h)$
while $|V \setminus T_h| > 0$ **do**
  $h := h + 1$
  Determine two adjacent vertices i_j and i_{j+1} from T_{h-1} such that $\mathrm{w}(i_j, v_l) + \mathrm{w}(v_l, i_{j+1}) - \mathrm{w}(i_j, i_{j+1})$ is the minimum possible, where $v_l \in V \setminus T_{h-1}$
  $T_h :=$ insert vertex v_l in tour T_{h-1} between vertices i_j and i_{j+1}.
$T := T_h$
return T

Algorithm 4: $JOINP(T_{h-1}, P_h)$.

$T_{h-1}^0 := T_{h-1}$
for $l := 1$ **to** $|P_h|$ **do**
  Determine two adjacent vertices i_j and i_{j+1} from T_{h-1}^{l-1} such that $\mathrm{w}(i_j, v_l) + \mathrm{w}(v_l, i_{j+1}) - \mathrm{w}(i_j, i_{j+1})$ is the minimum possible.
  $T_{h-1}^l :=$ insert vertex v_l in tour T_{h-1}^{l-1} between vertices i_j and i_{j+1}.
$T_h := T_{h-1}^{|P_h|}$
return T_h

Algorithm 5: $INNER\_CONVEX\_HULL(V, T_{h-1}, c_h)$.

if $|V \setminus T_{h-1}| \geq 3$ **then**
  Calculate the angles $\theta(c_h, v_i)$; $\forall\, v_i \in V \setminus T_{h-1}$
  $l := 0$
  $L_h^0 := (v_1, v_2, \cdots, v_{|V \setminus T_{h-1}|})$; such that $\theta(c_h, v_1) \geq \theta(c_h, v_2) \geq \cdots \geq \theta(c_h, v_{|V \setminus T_{h-1}|})$.
  $j := 1$
  while $j \leq |L_h^l| \ \wedge\ |L_h^l| > 3$ **do**
    $l := l + 1$
    // if $v_j = v_1$, then $v_{j-1} = v_{|L_h^{l-1}|}$, and if $v_j = v_{|L_h^{l-1}|}$, then $v_{j+1} = v_1$
    if $\phi((v_{j-1}, v_j, v_{j+1}), c_h) > \pi$ **then**
      if $\mathrm{w}(c_h, v_{j-1}) > \mathrm{w}(c_h, v_{j+1})$ **then**
        $L_h^l := L_h^{l-1} \setminus \{v_{j-1}\}$
        $j := j - 1$
      else
        $L_h^l := L_h^{l-1} \setminus \{v_{j+1}\}$
    else
      $j := j + 1$
  $P_h := L_h^l$
  return P_h
else
  "It is not possible to construct a polygon with $|V \setminus T_{h-1}| < 3$ vertices."

2.4.1. Algorithm 5, ($INNER\_CONVEX\_HULL$) for the Construction of Polygon P_h.

Given an external iteration h in Algorithm 3 ($INFLAMMATION$), we denote by L_h^l a *polygonal line* of the iteration l in Algorithm 5 (Figure 13), ($INNER\_CONVEX\_HULL$), that is a broken line of a finite number of edges formed by some vertices from set $V \setminus T_{h-1}$. Polygonal line T_h serves an auxiliary purpose at every iteration h with $|V \setminus T_{h-1}| \geq 3$ in Algorithm 3.

Consider an iteration h with $|V \setminus T_{h-1}| \geq 3$ in Algorithm 3 . Initially, at iteration 0 in Algorithm 5, a polygonal line L_h^0 consists of all vertices of set $V \setminus T_{h-1}$ sorted in non-increasing order of their angles $\theta(c_h, v_i)$ (see Equation (4)) (i.e., $L_h^0 = (v_1, v_2, \cdots, v_{|V \setminus T_{h-1}|})$ where $\theta(c_h, v_1) \geq \theta(c_h, v_2) \geq \cdots \geq \theta(c_h, b_{|V \setminus T_{h-1}|})$) (see Figure 14).

Consider the polygonal line with an additional edge $(v_1, v_{|V \setminus T_{h-1}|})$ joining the two extremal vertices v_1 and $v_{|V \setminus T_{h-1}|})$ of the polygonal line L_h^0. In case the polygonal line obtained in this way forms a convex polygon, then the algorithm delivers this polygon P_h.

Otherwise, at iteration $l > 0$, the polygon P_h is obtained from the polygonal line L_h^{l-1} by eliminating some of its vertices and the two edges associated with every eliminated vertex. Instead of the two eliminated edges, one additional edge is created. This is accomplished by an iterative analysis of three consecutive vertices (v_{j-1}, v_j, v_{j+1}) from the polygonal line L_h^{l-1} at once, where $v_{j-1} = v_{|L_h^{l-1}|}$ if $v_j = v_1$ and $v_{j+1} = v_1$ if $v_j = v_{|L_h^{l-1}|}$), for every $j = 1, \ldots, |L_h^{l-1}|$. For such a trio of consecutive points, we define a new angle-related function $\phi(v_{j-1}, v_j, v_{j+1}, c_h)$; see Equation (5) below. It returns the angle between the edges (v_{j-1}, v_j) and (v_j, v_{j+1}) in the range $[-\pi, \pi]$; see Figure 15.

$$\begin{aligned} \phi_1(v_{j-1}, v_j, c_h) &= \arccos\left(\frac{\mathrm{w}(c_h, v_{j-1})^2 - \mathrm{w}(c_h, v_j)^2 - \mathrm{w}(v_{j-1}, v_j)^2}{-2\,\mathrm{w}(c_h, v_j)\,\mathrm{w}(v_{j-1}, v_j)}\right) \\ \phi_2(v_j, v_{j+1}, c_h) &= \arccos\left(\frac{\mathrm{w}(c_h, v_{j+1})^2 - \mathrm{w}(c_h, v_j)^2 - \mathrm{w}(v_j, v_{j+1})^2}{-2\,\mathrm{w}(c_h, v_j)\,\mathrm{w}(v_j, v_{j+1})}\right) \\ \phi(v_{j-1}, v_j, v_{j+1}, c_h) &= \phi_1(v_{j-1}, v_j, c_h) + \phi_2(v_j, v_{j+1}, c_h) \end{aligned} \tag{5}$$

Given a trio (v_{j-1}, v_j, v_{j+1}) of iteration l, the algorithm verifies if the internal angle between the edges (v_{j-1}, v_j) and (v_j, v_{j+1}), $\phi((v_{j-1}, v_j, v_{j+1}), c_h)$, is no larger than π radians. If this holds, then all three points (v_{j-1}, v_j, v_{j+1}) are kept in polygon P_h, i.e., $P_h := L_h^{l-1}$. Otherwise, the polygonal path is non-convex; hence, one of the points is to be removed: If $\mathrm{w}(c_h, v_{j-1}) > \mathrm{w}(c_h, v_{j+1})$, then the vertex v_{j-1} is removed, i.e., $L_h^l := L_h^{l-1} \setminus \{v_{j-1}\}$. Otherwise, the vertex v_{j+1} is removed, i.e., $L_h^l := L_h^{l-1} \setminus \{v_{j+1}\}$. Let l be the last iteration.

We illustrate the construction process of the polygon line L_1^0 for problem instance "tsp10". In Figure 14, $T_{h-1} = \emptyset$ and the centroid $c_h = c_1$ with coordinates $(52.8, 55)$ is depicted in red color. Figure 14a illustrates how the θ angles are calculated so that $\theta(c_1, 4) \geq \theta(c_1, 10) \geq \theta(c_1, 6) \geq \theta(c_1, 10) \geq \theta(c_1, 1) \geq \theta(c_1, 5) \geq \theta(c_1, 2) \geq \theta(c_1, 3) \geq \theta(c_1, 8) \geq \theta(c_1, 9) \geq \theta(c_1, 7)$. Figure 14b shows the resultant polygonal line $L_1^0 = (4, 10, 6, 1, 5, 2, 3, 8, 9, 7)$.

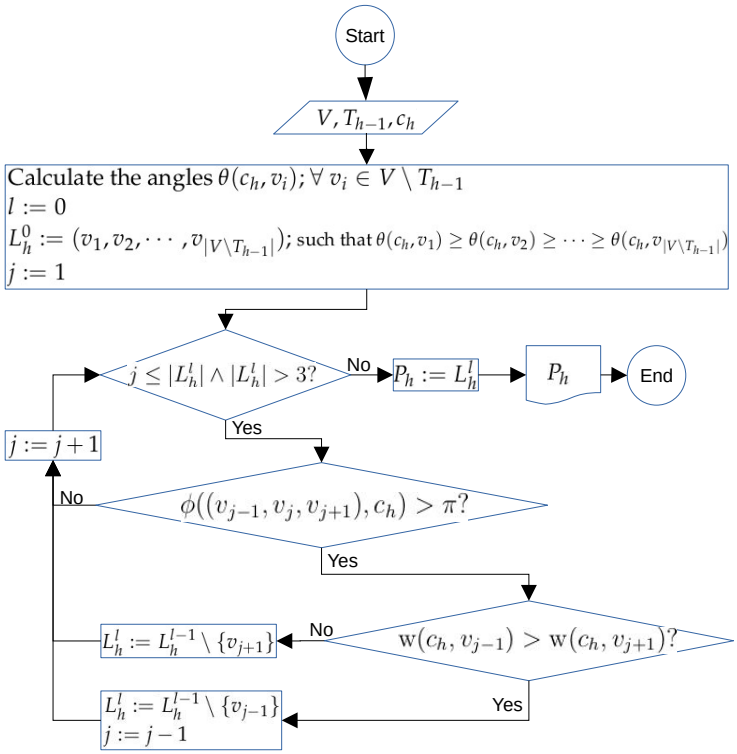


Figure 13. Inner convex hull algorithm flowchart.

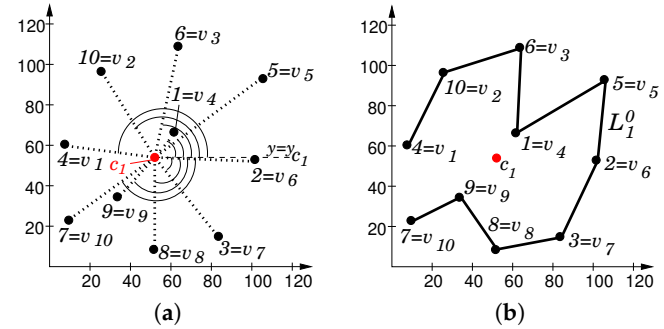


Figure 14. Construction of the initial polygonal line L_1^0 for the example instance "tsp10", where $c_1 = (52.8, 55)$.

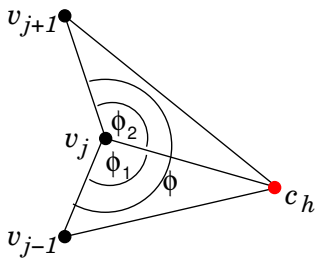


Figure 15. Angle $\phi(v_{j-1}, v_j, v_{j+1}, c_h)$ formed by the edges (v_{j-1}, v_l) and (v_j, v_{j+1}), where v_{j-1}, v_j, and v_{j+1} are consecutive vertices from polygonal line L_h^{l-1}.

Table 2 and Figure 16 show how Algorithm 5 ($INNER\_CONVEX\_HULL$) constructs the polygon P_1 for problem instance "tsp10". Figure 16a illustrates the initial polygonal line $L_1^1 = (4, 10, 6, 1, 5, 2, 3, 8, 9, 7)$. The first internal angle found in L_1^3 that is greater than π radians is between the edges $(6, 1)$ and $(1, 5)$. Since $w(c_1, 6) < w(c_1, 5)$, vertex 5 is removed from L_1^3. In Figure 16b, the angle between the edges $(6, 1)$ and $(1, 2)$ is greater than π

radians. Since $w(c_1,6) > w(c_1,2)$, vertex 2 is removed from L_1^4. In Figure 16c, the angle between the edges $(10,1)$ and $(1,2)$ is greater than π radians. Since $w(c_1,10) > w(c_1,2)$, vertex 10 is removed from L_1^5. In Figure 16d, the angle between the edges $(8,9)$ and $(9,7)$ is greater than π radians. Since $w(c_1,8) < w(c_1,7)$, vertex 7 is removed from L_1^{10}. In Figure 16e, the angle between the edges $(8,9)$ and $(9,4)$ is greater than π radians. Since $w(c_1,8) > w(c_1,4)$, vertex 8 is removed from L_1^{11}. Figure 16f shows that the resultant inner convex hull $P = (4,1,2,3,9)$ is delivered.

Table 2. Iterations performed by Algorithm 5 (*INNER_CONVEX_HULL*) for the construction of the inner convex polygon P for the example instance "tsp10".

l	L_h^l	j	v_j	*Condition*	*Decision*
0	Polygonal line is constructed $L_1^0 := (4,10,6,1,5,2,3,8,9,7)$				
1	$(4,10,6,1,5,2,3,8,9,7)$	1	4	$\phi(7,4,10,c) \leq \pi$: *True*	$L_1^1 := L_1^0, j := j+1$
2	$(4,10,6,1,5,2,3,8,9,7)$	2	10	$\phi(4,10,6,c) \leq \pi$: *True*	$L_1^2 := L_1^1, j := j+1$
3	$(4,10,6,1,5,2,3,8,9,7)$	3	6	$\phi(10,6,1,c) \leq \pi$: *True*	$L_1^3 := L_1^2, j := j+1$
4	$(4,10,6,1,5,2,3,8,9,7)$	4	1	$\phi(6,1,5,c) \leq \pi$: *False*	$L_1^4 := L_1^3 \setminus \{5\}$
5	$(4,10,6,1,2,3,8,9,7)$	4	1	$\phi(6,1,2,c) \leq \pi$: *False*	$L_1^5 := L_1^4 \setminus \{6\}, j := j-1$
6	$(4,10,1,2,3,8,9,7)$	3	1	$\phi(10,1,2,c) \leq \pi$: *False*	$L_1^6 := L_1^5 \setminus \{10\}, j := j-1$
7	$(4,1,2,3,8,9,7)$	2	1	$\phi(4,1,2,c) \leq \pi$: *True*	$L_1^7 := L_1^6, j := j+1$
8	$(4,1,2,3,8,9,7)$	3	2	$\phi(1,2,3,c) \leq \pi$: *True*	$L_1^8 := L_1^7, j := j+1$
9	$(4,1,2,3,8,9,7)$	4	3	$\phi(2,3,8,c) \leq \pi$: *True*	$L_1^9 := L_1^8, j := j+1$
10	$(4,1,2,3,8,9,7)$	5	8	$\phi(3,8,9,c) \leq \pi$: *True*	$L_1^{10} := L_1^9, j := j+1$
11	$(4,1,2,3,8,9,7)$	6	9	$\phi(8,9,7,c) \leq \pi$: *False*	$L_1^{11} := L_1^{10} \setminus \{7\}$
12	$(4,1,2,3,8,9)$	6	9	$\phi(8,9,4,c) \leq \pi$: *False*	$L_1^{12} := L_1^{11} \setminus \{8\}, j := j-1$
13	$(4,1,2,3,9)$	5	9	$\phi(3,9,4,c) \leq \pi$: *True*	$L_1^{13} := L_1^{12}, j := j+1$
14	Inner convex hull $P_1 := L_1^{13} = (4,1,2,3,9)$ is constructed.				

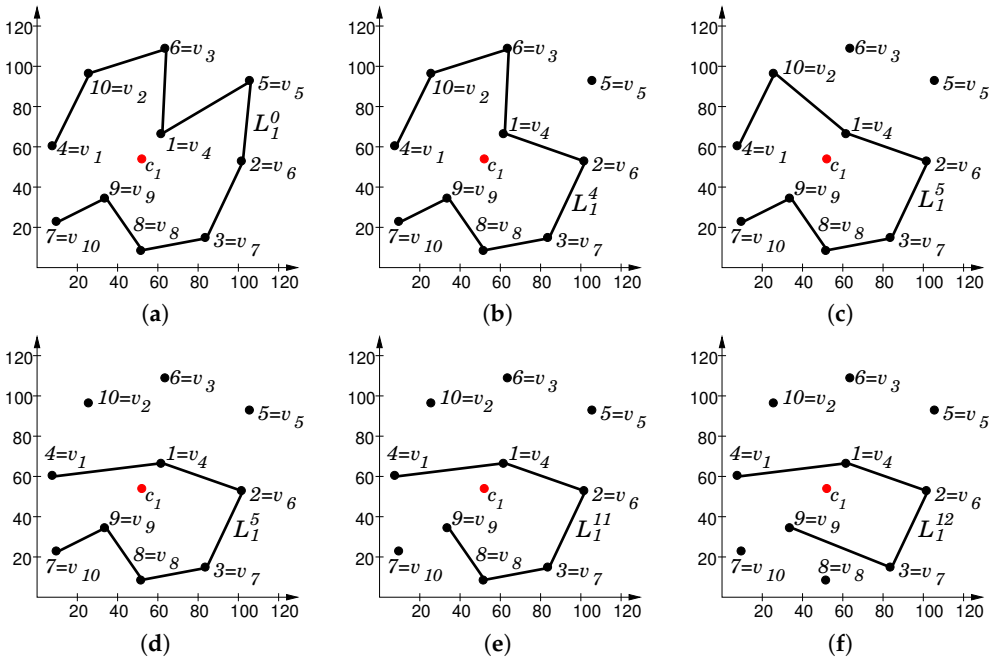


Figure 16. Graphs corresponding to the iterations carried out by Algorithm 5 (*INNER_CONVEX_HULL*) for the construction of the inner convex hull P for the example instance "tsp10". (**a**) $L_1^3 := L_1^2 := L_1^1 := L_1^0$; (**b**) $L_1^4 := L_1^3 \setminus \{5\}$; (**c**) $L_1^5 := L_1^4 \setminus \{6\}$; (**d**) $L_1^{10} := L_1^9 := L_1^8 := L_1^7 := L_1^6 := L_1^5 \setminus \{10\}$; (**e**) $L_1^{11} := L_1^{10} \setminus \{7\}$; (**f**) $L_1^{13} := L_1^{12} := L_1^{11} \setminus \{8\}$.

2.4.2. Algorithm JOINP

Algorithm 4 (Figure 17) (*JOINP*) is invoked at every iteration h from Algorithm 3 (*INFLAMMATION*), where Algorithm 5 ($INNER\_CONVEX\_HULL$) is invoked before returning polygon P_h. Algorithm 4 constructs tour T_h based on an earlier-constructed tour $T_{h-1} = (i_1, \ldots, i_{|T_{h-1}|})$ and polygon $P_h = (v_1, \ldots, v_{|P_h|})$. The construction is carried out in a number of iterations, where we denote by T^l_{h-1} a partial tour constructed at iteration l; we let $T^0_{h-1} := T_{h-1}$.

At iteration $l \geq 1$, vertex v_l from polygon P_h is inserted in between two adjacent pairs of vertices from tour T^l_{h-1}. These two vertices i_j and i_{j+1} are determined so that the cost of inserting vertex $v_l \in P_h$ in between these vertices is the minimum possible, i.e.,

$$\mathrm{w}(i_j, v_l) + \mathrm{w}(v_l, i_{j+1}) - \mathrm{w}(i_j, i_{j+1})$$

is the minimum possible (the triangle inequality). Below is a formal description of the algorithm.

Figure 18 shows tour T_1 and polygon P_2 for the problem instance "tsp10". Table 3 and Figure 19 show the five iterations performed by Algorithm 4 (*JOINP*) for the construction of tour T_2.

Table 3. Iterations performed by Algorithm 4 (*JOINP*) for the construction of tour T_2.

l	v_l	(i_j, i_{j+1})	T^l_{h-1}
0			$(4, 1, 2, 3, 9, 4)$
1	10	$(4, 1)$	$(4, 10, 1, 2, 3, 9, 4)$
2	6	$(10, 1)$	$(4, 10, 6, 1, 2, 3, 9, 4)$
3	5	$(1, 2)$	$(4, 10, 6, 1, 5, 2, 3, 9, 4)$
4	8	$(3, 9)$	$(4, 10, 6, 1, 5, 2, 3, 8, 9, 4)$
5	7	$(9, 4)$	$(4, 10, 6, 1, 5, 2, 3, 8, 9, 7, 4)$

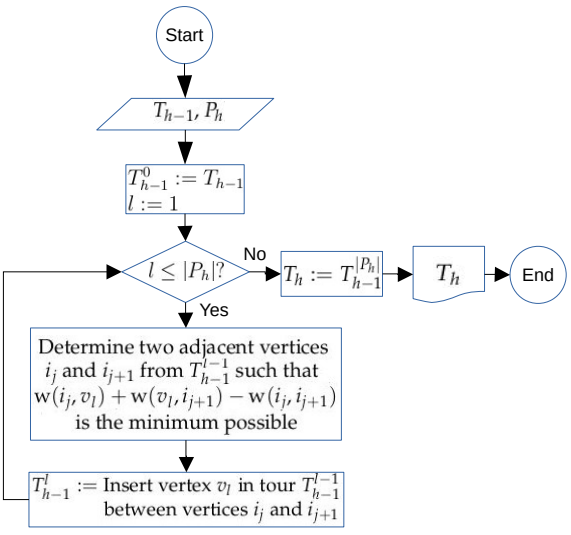


Figure 17. Joinp algorithm flowchart.

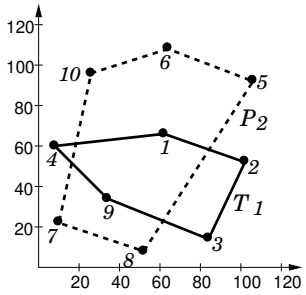


Figure 18. Tour T_1 and polygon P_2 for the example instance "tsp10".

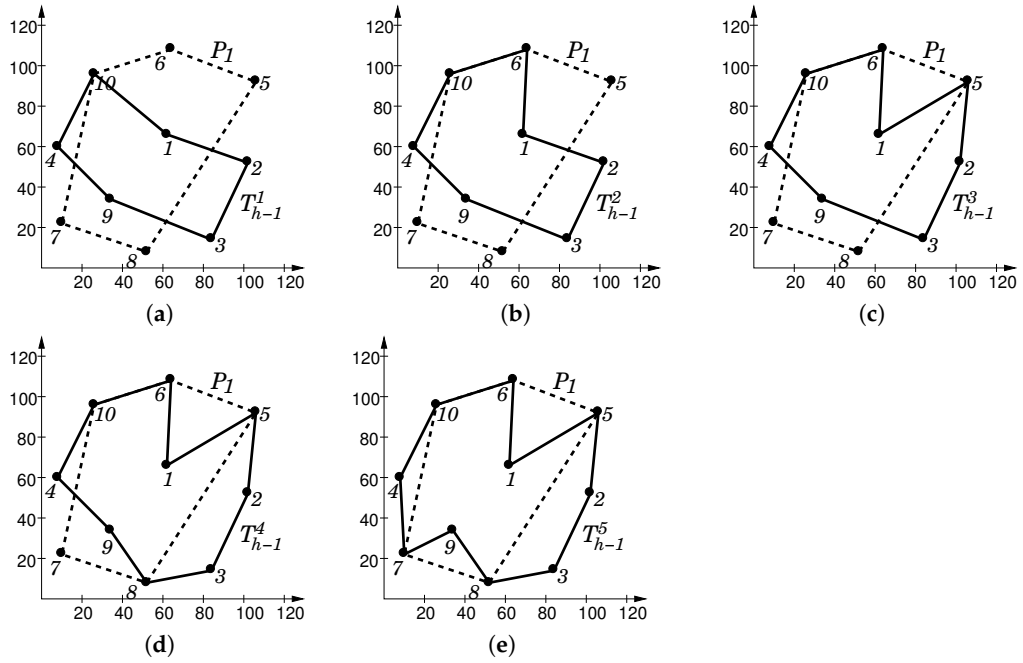


Figure 19. Graphs corresponding to the iterations carried out by Algorithm 4 ($JOINP$) for the construction of tour T_2. **(a)** $T^1_{h-1} = (4, 10, 1, 2, 3, 9, 4)$; **(b)** $T^2_{h-1} = (4, 10, 6, 1, 2, 3, 9, 4)$; **(c)** $T^3_{h-1} = (4, 10, 6, 1, 5, 2, 3, 9, 4)$; **(d)** $T^4_{h-1} = (4, 10, 6, 1, 5, 2, 3, 8, 9, 4)$; **(e)** $T_2 := T^5_{h-1} = (4, 10, 6, 1, 5, 2, 3, 8, 9, 7, 4)$.

2.4.3. Algorithm INFLAMMATION

The Algorithm 3 (Figure 20) initially lets $T_0 := \emptyset$. While $|V \setminus T_h| > 2$, in the iteration $h > 0$, the coordinates x_{c_h} and y_{c_h} of the centroid c_h are calculated as follows:

$$x_{c_h} = \frac{\sum_{i \in V \setminus T_{h-1}} x_i}{|V \setminus T_{h-1}|}$$

and

$$y_{c_h} = \frac{\sum_{i \in V \setminus T_{h-1}} y_i}{|V \setminus T_{h-1}|}.$$

Once the centroid is so defined, Algorithm 5 ($INNER\_CONVEX\_HULL$) is invoked to obtain the inner convex hull P_h. Then, the tour T_h is constructed by the invoking of Algorithm 4 ($JOINP(T_{h-1}, c_h)$).

If $|V \setminus T_h| = 0$, then the algorithm halts. Else, if $1 \leq |V \setminus T_h| \leq 2$, instead of invoking Algorithm 5 (no inner convex hull can be constructed), the $|V \setminus T_h|$ vertices from set $V \setminus T_h$ are inserted in between two consecutive vertices of tour T_{h-1} in $|V \setminus T_h|$ iterations, one vertex at each iteration, similar to as in Algorithm 4 (so that the increase in the total cost at each iteration is the minimum possible).

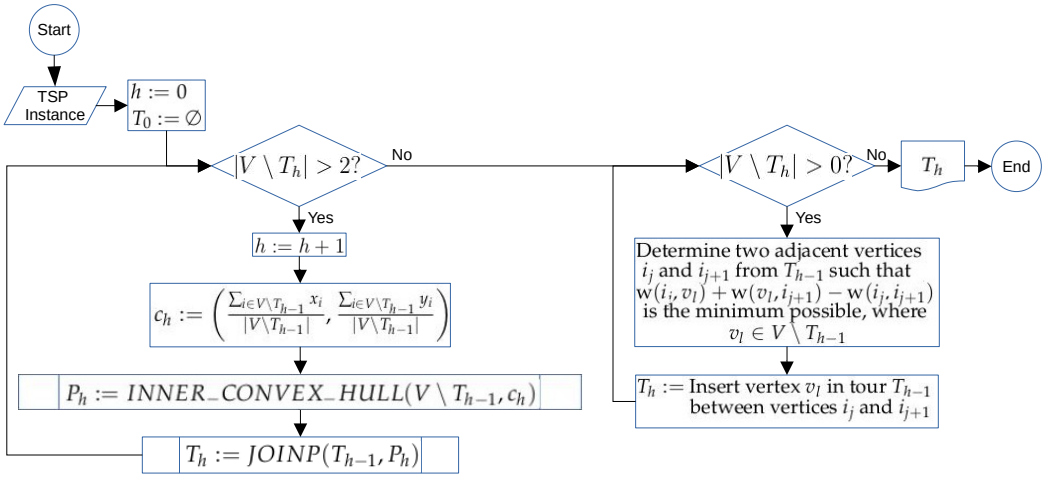


Figure 20. Inflammation algorithm flowchart.

Table 4 and Figure 21 illustrate Algorithm 3 for problem instance "tsp10" with 10 vertices. In Figure 21a, inner convex polygon $P_1 = (4, 1, 2, 3, 9)$ is constructed, considering the vertices of $V \setminus T_0 = V$. In Figure 21b, $T_1 = (4, 1, 2, 3, 9, 4)$ is constructed with the vertices included in P_1 and $P_2 = (7, 10, 6, 5, 8)$ is constructed with the vertices included in $V \setminus T_1$. In Figure 21c, $T_2 = (4, 10, 6, 1, 5, 2, 3, 8, 9, 7, 4)$ is constructed by inserting the vertices from P_2 in tour T_1. The resultant feasible solution $T = T_2$.

Table 4. Iterations performed by Algorithm 3 (*INFLAMMATION*) for the construction of tour T for example instance "tsp10".

h	P_h	T_h
0		$\emptyset$
1	$(4, 1, 2, 3, 9)$	$(4, 1, 2, 3, 9, 4)$
2	$(10, 6, 5, 8, 7)$	$(4, 10, 6, 1, 5, 2, 3, 8, 9, 7, 4)$
		$T = (4, 10, 6, 1, 5, 2, 3, 8, 9, 7, 4)$

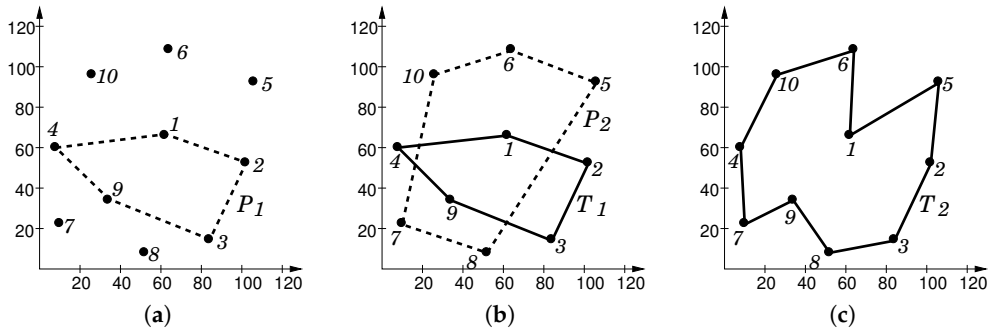


Figure 21. Graphs of the iterations performed by Algorithm 3 (*INFLAMMATION*) to solve the example instance "tsp10".

We applied an improvement algorithm to a complete tour obtained as above. Two heuristics were used in a loop for this purpose. First, a local search algorithm proposed by Croes ("*2-Opt*") [43] was applied. Then, iteratively, a vertex from the current tour was relocated to an alternative position if this yielded further reduction in the total cost. The algorithm halts when no such rearrangement exists.

As is easy to see, algorithm *INNER_CONVEX_HULL* has time complexity $O(n \log n)$; algorithm JOINP has time complexity $O(n^2)$; the improvement phase has the same time complexity as the improvement phase of the INSERTION algorithm.

3. Implementation and Results

In this section, we report our experimental study. Our algorithms were coded in C++ and compiled on a Debian GNU/Linux operating system.

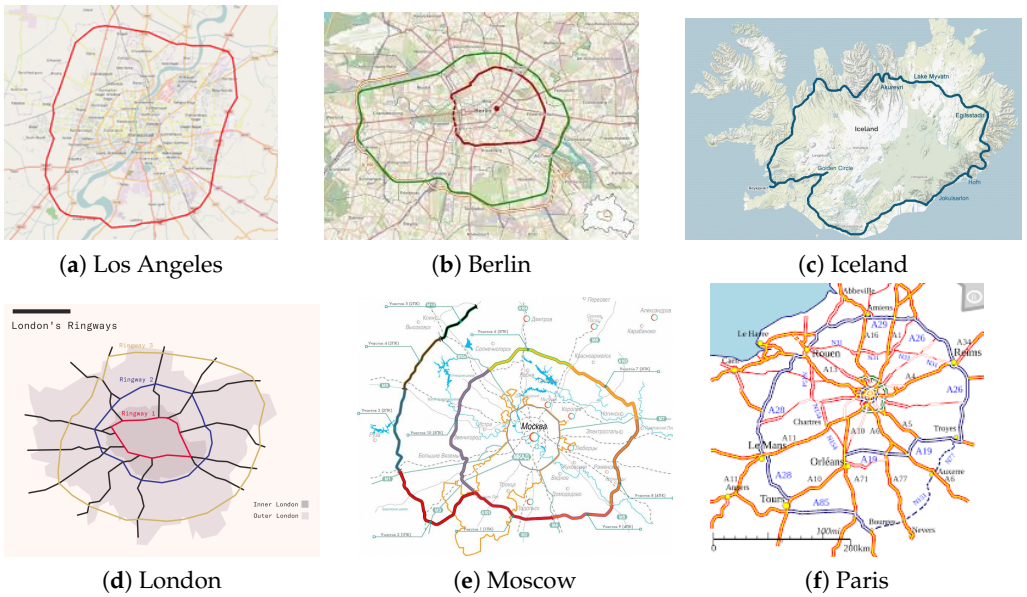
According to our earlier experimental results from [5], the insertion algorithm from Section 2.2 proved to be very fast. It was tested for 218 problem instances with sizes between 32 and 744,710 from the TSPLIB [44], NATIONAL [45], ART GALLERY [45], and VLSI [45] group of instances. On average, the cost of the approximation obtained by this method with respect to the cost of the best-known solution had a margin of error of 7.2%.

The inflammation method was designed to provide solutions of good quality for some typical real-life instances. A few such sample instances were created based on the existing tourist tours and network convenience stores from different parts of the world (Asia, Europe, South America, United States of America, and Mexico. We describe these instances in the following subsections.

The algorithm for the BMTSP from Section 2.3 was tested for 22 benchmark instances. For one of them, $pr1002.tsp$ (with $k = 5$, $m_{\min} = 1$, and $m_{\max} = 5$), we obtained the best-known result $\mathrm{C}(T) = 329,128$, improving the earlier-known best cost $\mathrm{C}(BKS) = 334.351$ Lo et al. [38]. The remaining 16 instances were solved with an approximation gap less than or equal to 7%. We refer the reader to Table 1 from [42] for the details.

3.1. Ring Roads

Many towns throughout the world have an important principal avenue, often referred to as ring (circular) roads or peripheral avenues. They typically surround the populated areas. They can be seen as circle-type curves or multiple-sided polygons that contain a given populated area in the interior area. Peripheral avenues contain different important locations such as military barracks, supermarkets, malls with cinemas, restaurants, gas stations, etc. The remaining interior destinations such as market places, bus terminals, teacher training colleges, fields for physical and sports activities, houses, etc., can also be reached through peripheral avenues. Peripheral avenues have also exits to highways that connect a given city with neighboring towns. In Figure 22, the peripheral roads of some major cities are illustrated (these images were taken from the Internet). Based on these images, we created our first sample problem instance for an abstract peripheral avenue and tested the inflammation algorithm for this instance.

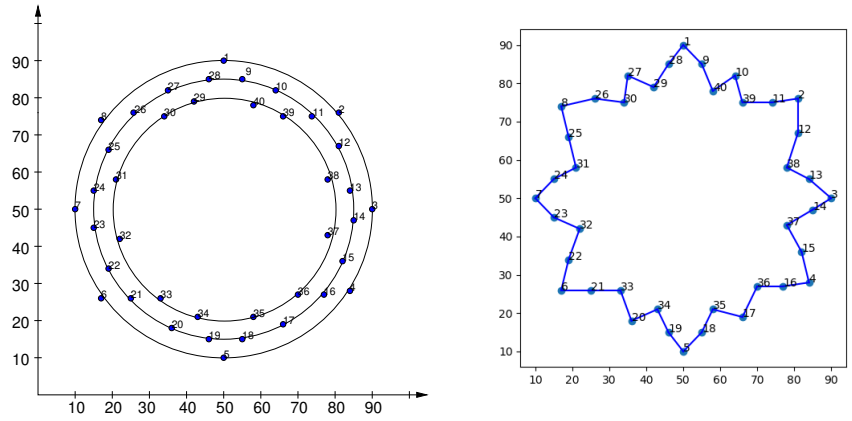


(**a**) Los Angeles (**b**) Berlin (**c**) Iceland

(**d**) London (**e**) Moscow (**f**) Paris

Figure 22. Peripheral roads of some major cities.

Figure 23a illustrates this sample instance, and Figure 23b represents the solution created by the algorithm. The optimality of the created tour is easily verified. For the real-life instances, strictly optimal solutions may not be obtained, but one would normally expect the algorithm to deliver close to the optimal solutions. In the following subsections, we give the results for six other real-life instances.



(**a**) Instance (**b**) Solution

Figure 23. Instance "*FNV40.tsp*" and its solution.

3.2. Convenience Stores in Iguala

Our next instance was taken from a map of the city of Iguala, located in the state of Guerrero, Mexico (See Appendix A.1). Different important spots including a chain of convenience stores are located around the peripheral avenue of this city. This peripheral avenue is shaped as an eight-sided polygon, as we can see from Figure 24. In Iguala, an important chain of convenience stores, Oxxo, has 21 stores and an administrative office in this city; 11 of these stores are located on the peripheral avenue or one block from it; eight stores are in the city center; two stores are at one of the city exits. Among the employees who work in those offices, there are those who must visit all the stores, and there are also a couple of supervisors who only visit between 10 and 13 stores that they are in charge of.

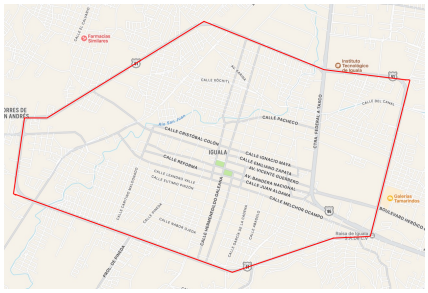


Figure 24. The peripheral avenue of Iguala is shaped as an 8-sided polygon [46].

We tested both the inflammation and BMTSP algorithms for this instance. Figure 25 shows the iterations carried out by the inflammation method for the construction of the tour for an employee to visit all the stores in the city. The vertex d represents the administrative office from which that employee starts and ends his/her trip.

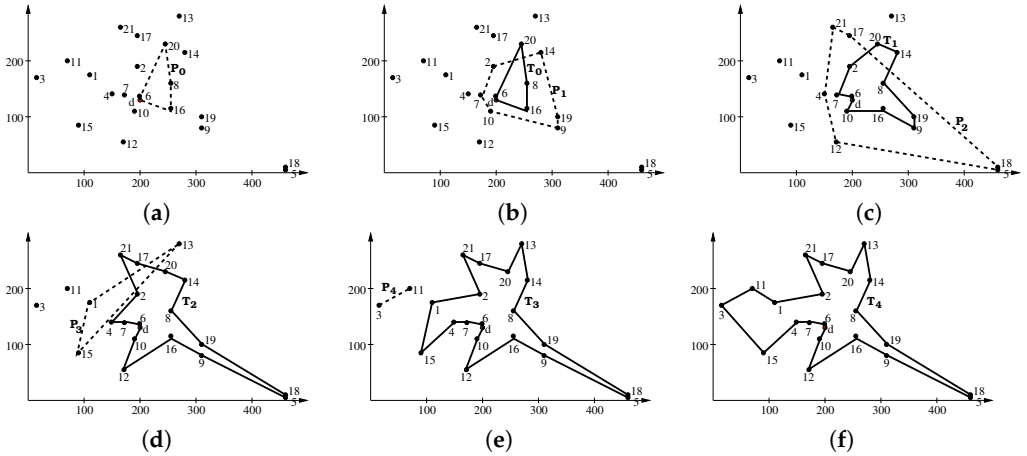


Figure 25. Iterations performed to resolve the instance of Iguala by the inflammation method.

In Figure 25a, the polygon $P_0 = (d, 6, 20, 8, 16)$ is constructed. In Figure 25b, $T_0 = (d, 6, 20, 8, 16, d)$ is constructed from the vertices of P_0, and the polygon $P_1 = (7, 2, 14, 19, 9, 10)$ is constructed. In Figure 25c, $T_1 = (d, 6, 7, 2, 20, 14, 8, 19, 9, 16, 10, d)$ is constructed from the vertices of P_1 inserted in T_0, and the polygon $P_2 = (4, 21, 17, 18, 5, 12)$ is constructed. In Figure 25d, $T_2 = (d, 6, 7, 4, 2, 21, 17, 20, 14, 8, 19, 18, 5, 9, 16, 12, 10, d)$ is constructed from the vertices of P_2 inserted in T_1, and the polygon $P_3 = (1, 13, 15)$ is constructed. In Figure 25e, $T_3 = (d, 6, 7, 4, 15, 1, 2, 21, 17, 20, 13, 14, 8, 19, 18, 5, 9, 16, 12, 10, d)$ is constructed from the vertices of P_3 inserted in T_2, and the polygon $P_4 = (3, 11)$ is constructed. In Figure 25f, $T_4 = (d, 6, 7, 4, 15, 3, 11, 1, 2, 21, 17, 20, 13, 14, 8, 19, 18, 5, 9, 16, 12, 10, d)$ is constructed from the vertices of P_4 inserted in T_3, and the method completes the construction of the tour.

Figure 26 shows the phases carried out by the bounded multiprocessor TSP method for the construction of a tour for each of the two supervisors (with $k = 2$, $m_{\min} = 10$ and $m_{\max} = 13$). In Figure 26a, the partition of set $V \setminus \{d\}$ into two feasible subsets $V_1 = \{5, 8, 9, 13, 14, 16, 17, 18, 19, 20, 21\}$ and $V_2 = \{1, 2, 3, 4, 6, 7, 10, 11, 12, 15\}$ is shown, whose number of vertices included in each of them is between 10 and 13 vertices. In Figure 26b, tours $T^1 = (d, 21, 17, 20, 13, 14, 8, 8, 5, 18, 19, 16, d)$ and $T^2 = (d, 6, 2, 7, 4, 1, 11, 3, 15, 12, 10, d)$ are constructed for the subsets $V_1 \cup \{d\}$ and $V_2 \cup \{d\}$, respectively. In Figure 26c, the feasible solution constructed is improved.

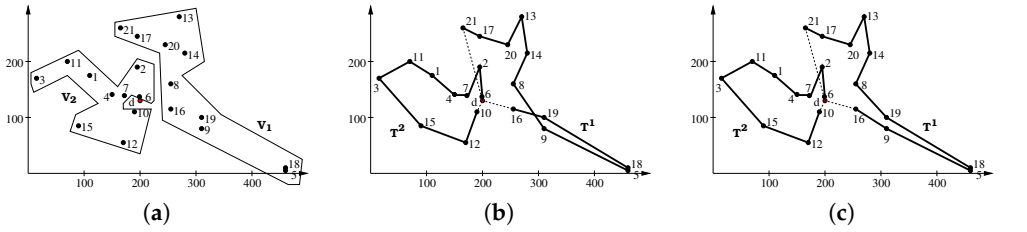
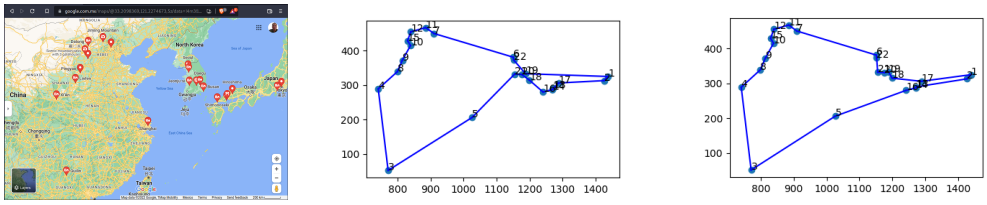


Figure 26. Tours T^1 y T^2 constructed by the bounded multiprocessor TSP method for 2 supervisors, $m_{\min} = 10$ and $m_{\max} = 13$.

3.2.1. Instance Asia

Instance "*Asia.tsp*" was built based on two tourist circuits: "Japan, China and South Korea in 15 days" by [47] and "17 days—Travel to Japan, South Korea and China" by [48].

The instance includes 22 tourist places: Tokyo, Hakone, Guilin, Xi'an, Shanghai, Seoul, Beijing, Linfen, Pingyao, Wutai, Jiming Mountain, Datong, Itsukushima Shrine, Iwakuni, Xinzhou, Shimonoseki, Hiroshima, Busan, Daegu, Tripitaka, Jeonju, and Suwon (see Figure 27).



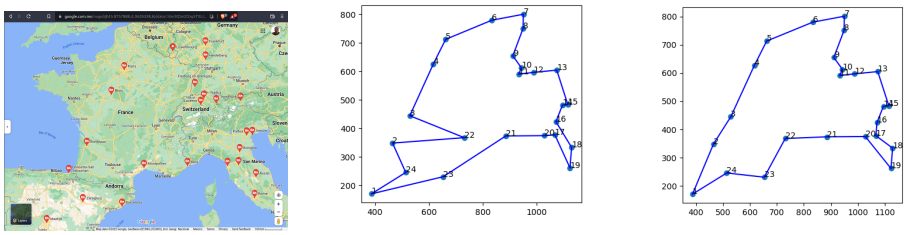
(**a**) (**b**) (**c**)

Figure 27. Map and solution of instance "*Asia.tsp*". (**a**) Map [49]; (**b**) Solution; (**c**) Improved solution.

3.2.2. Instance Europe

Instance "*Europe.tsp*" was taken from the circuit called "Fantastic Europe Circuit" from [50].

The instance includes 24 tourist places: Madrid, San Sebastian, Bordeaux, Blois, Paris, Luxembourg, Frankfurt, Heidelberg, Freiburg, Zurich, Lucerne, Vaduz, Innsbruck, Padova, Venice, Bologna, Florence, Assisi, Rome, Pisa, Nice, Montpellier, Barcelona, and Zaragoza (see Figure 28).



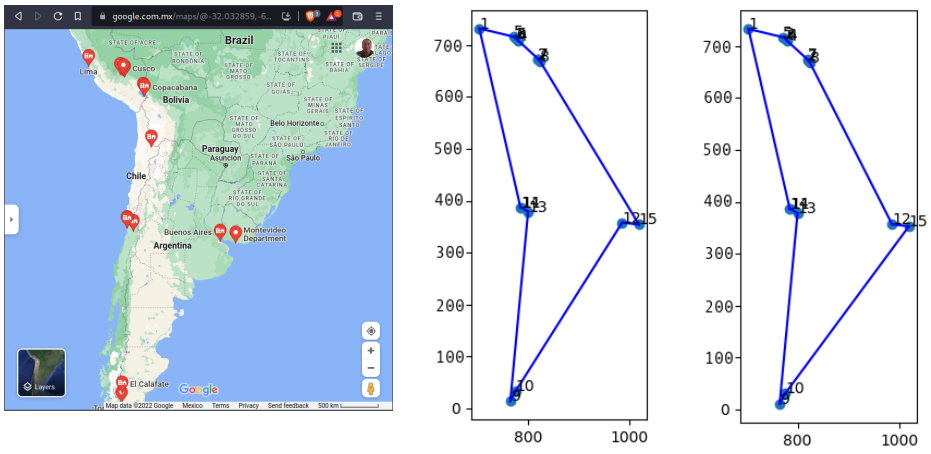
(**a**) (**b**) (**c**)

Figure 28. Map and solution of instance "*Europe.tsp*". (**a**) Map [51]; (**b**) Solution; (**c**) Improved solution.

3.2.3. Instance South America

Instance "*SouthAmerica.tsp*" was built based on two tourist circuits: "Peru, Bolivia and Chile" and "Argentina-Chile" of [52].

The instance includes 15 tourist places: Lima, San Pedro Atacama, Ollantaytambo, Cusco, Machu Picchu, Sacred Valley of the Incas, Lake Titicaca, Copacabana, Torres de Paine, El Calafate, Viña del Mar, Buenos Aires, Santiago de Chile, Valparaíso, and Montevideo (See Figure 29).



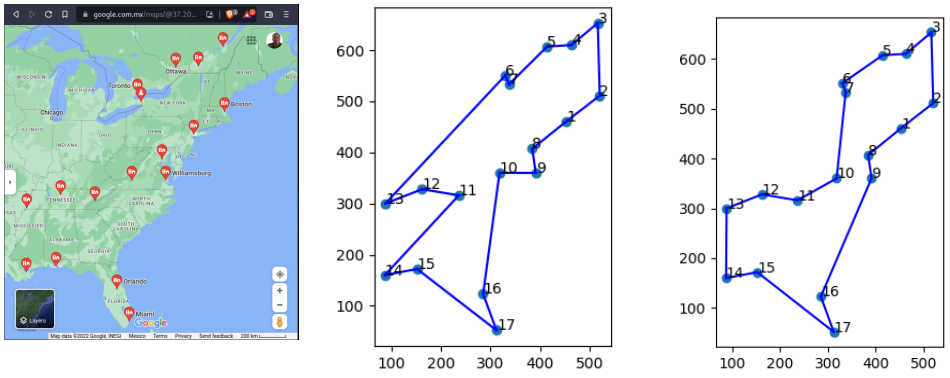
(**a**) (**b**) (**c**)

Figure 29. Map and solution of instance "*SouthAmerica.tsp*". (**a**) Map [53]; (**b**) Solution; (**c**) Improved solution.

3.2.4. Instance East of USA

Instance "*EastUSA.tsp*" was taken from the circuit called "The Grand East" of [54].

The instance includes 19 tourist places: New York, Boston, Quebec, Montreal, Ottawa, Toronto, Niagara Falls, Amish Country, Washington D.C., Wiliamsburg, Roanoke, Gatlingburg, Nashville, Memphis, New Orleans, Pensacola, Orlando, and Miami (See Figure 30).



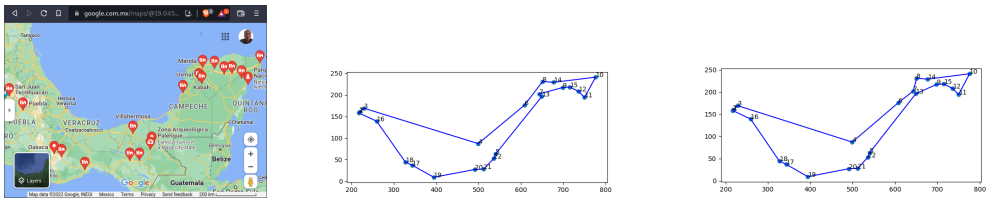
(**a**) (**b**) (**c**)

Figure 30. Map and solution of instance "*EastUSA.tsp*". (**a**) Map [55]; (**b**) Solution; (**c**) Improved solution.

3.2.5. Instance Mexico

The instance "*Mexico.tsp*" was built based on three tourist circuits: "Mayan Civilizations" and "Yucatán" from [56] and "The essentials of Mexico to Cancun " from [57].

The instance includes 22 tourist places: Mexico City, Teotihuacan, Villahermosa, Palenque, Campeche, Uxmal, Mérida, Chichen Itzá, Cancún, Tulum, Cobá, Kabah, Izamal, Valladolid, Puebla, Mitla, Oaxaca-MonteAlbán, Tehuantepec, Tuxtla Gutierrez, San Cristobal de las Casas, and Agua Azul Waterfalls (see Figure 31).



(**a**) (**b**) (**c**)

Figure 31. Map and solution of instance "*Mexico.tsp*". (**a**) Map [58]; (**b**) Solution; (**c**) Improved solution.

3.3. Summary of the Results at the Improvement Stage

In Table 5, we give a summary of the improvements accomplished at the last improvement stage of the inflammation algorithm. Column "$C_{Inf}(T)$" represents the cost of the tour delivered by the inflammation algorithm, and Column "$C_{Imp}(T)$" represents the cost of the tour delivered by the improvement stage. Column "% Improvement" presents the percentage of the improvement.

Table 5. A summary of the improvements accomplished at the last improvement stage of the inflammation algorithm.

Instance	$C_{Inf}(T)$	$C_{Imp}(T)$	% Improvement
FNV40.tsp	304.10	304.10	0%
Iguala.tsp	1490.78	1452.21	2.66 %
Asia.tsp	1878.60	1848.94	1.60%
Europe.tsp	2964.18	2578.17	14.97%
SouthAmerica.tsp	1690.70	1686.56	0.25%
EastUSA.tsp	1977.06	1761.28	12.25 %
Mexico.tsp	1389.89	1381.58	0.60 %

4. Conclusions and Future Work

Using convex polygons in our approach provided good-quality solutions for Euclidean traveling salesman problems. In particular, the inflammation algorithm provided optimal and near-optimal solutions for the "ring-type" instances of the Euclidean TSP. Ring-type instances with a symmetric geometric structure were solved optimally, whereas for the real-life ring-type instances, close to optimum solutions were created. As for the future work, it would be interesting to compare the practical behavior of the earlier-mentioned algorithms (including our algorithm) for the construction of a girding polygon. Indeed, the worst-case time complexity does not always reflect the practical performance. We also intend to extend the inflammation algorithm with a decomposition stage, which would partition a given problem instance into sub-instances with a nearly symmetric geometric structure. Then, we can apply the proposed algorithm to each of the derived substances and unify the created partial solutions into an overall complete solution. Likewise, our algorithm for the BMTSP can be extended with different alternative algorithms for partitioning the given set of points. Our method can also be adapted for more general settings including different versions of the multiprocessor traveling salesman problem.

Author Contributions: Conceptualization, N.V.; methodology, N.V. and V.H.P.-V.; validation, N.V.; formal analysis, N.V.; investigation, N.V., V.H.P.-V. and F.Á.H.-M.; Resource administration and Funding, J.A.H.-A.; writing—original draft preparation, V.H.P.-V. and N.V.; writing—review and editing, N.V.; visualization, V.H.P.-V. and J.A.H.-A.; supervision, N.V.; project administration, N.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research is expected to receive PRODEP funding from the Mexican Ministry of Superior Education (SEP)

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Appendix A.1

Figure A1 contains the location of the administrative office and the 21 convenience stores installed in Iguala, which are represented in Table A1, which contains the store number in column i, the x coordinate of store i in column x_i, and the y coordinate of store i in column y_i.

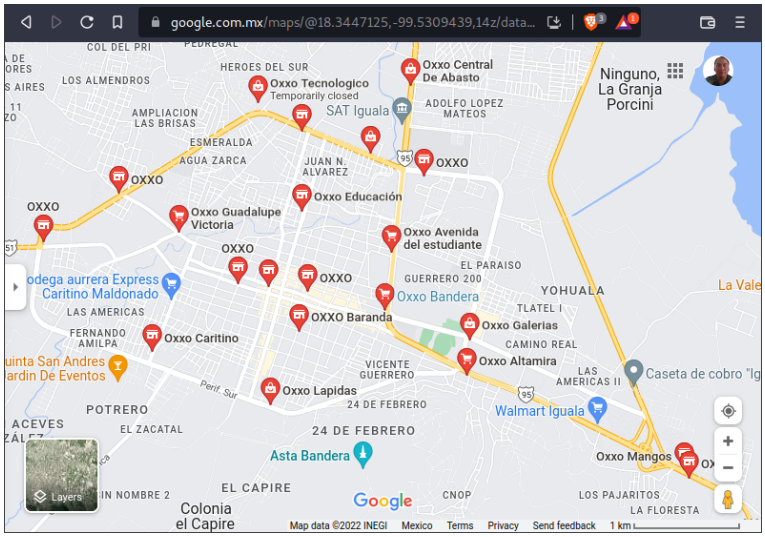


Figure A1. Location of the 21 convenience stores in Iguala. [46]

Table A1. Instance Iguala.

i	x_i	y_i	i	x_i	y_i
0	199	130	11	69	201
1	109	175	12	172	57
2	196	190	13	270	278
3	16	170	14	278	215
4	150	141	15	91	85
5	462	7	16	257	114
6	199	137	17	195	246
7	172	139	18	459	12
8	254	162	19	308	102
9	309	78	20	244	231
10	191	108	21	164	260

References

1. Karp, R.M. Reducibility among combinatorial problems. In *Complexity of Computer Computations*; Miller, R.E., Thatcher, J.W., Bohlinger, J.D., Eds.; Springer: Boston, MA, USA, 1972; pp. 85–103. [CrossRef]
2. Roerty, D.F. M-Salesman Balanced Tours Traveling Salesman Problem with Multiple Visits to Cities Allowed. Ph.D. Thesis, Texas Tech University, Lubbock, TX, USA, 1974. Available online: https://ttu-ir.tdl.org/bitstream/handle/2346/18193/31295002299211.pdf?sequence=1 (accessed on 19 August 2022).
3. Necula, R.; Breaban, M.; Raschip, M. Performance evaluation of ant colony systems for the single-depot multiple traveling salesman problem. In *International Conference on Hybrid Artificial Intelligence Systems*; Springer: Cham, Switzerland, 2015; pp. 257–268._22. [CrossRef]
4. Papadimitriou, C.H. The Euclidean Traveling Salesman Problem is NP-Complete. *Theor. Comput. Sci.* **1977**, *4*, 237–244. [CrossRef]
5. Pacheco-Valencia, V.; Hernández, J.A.; Sigarreta, J.M.; Vakhania, N. Simple Constructive, Insertion, and Improvement Heuristics Based on the Girding Polygon for the Euclidean Traveling Salesman Problem. *Algorithms* **2020**, *13*, 5. [CrossRef]
6. Vakhania, N.; Hernandez, J.A.; Alonso-Pecina, F.; Zavala, C. A Simple Heuristic for Basic Vehicle Routing Problem. *J. Comput. Sci. Technol. Updat.* **2016**, *3*, 38–44. Available online: https://cosmosscholars.com/phms/index.php/jcstu/article/view/693 (accessed on 13 July 2021). [CrossRef]
7. Macgregor, J.N.; Ormerod, T. Human performance on the traveling salesman problem. *Percept. Psychophys.* **1996**, *58*, 527–539. [CrossRef]
8. Norback, J.P.; Love, R.F. Geometric Approaches to Solving the Traveling Salesman Problem. *Manag. Sci.* **1977**, *23*, 1208–1223. [CrossRef]
9. Golden, B.; Bodin, L.; Doyle, T.; Stewart, W., Jr. Approximate Traveling Salesman Algorithms. *Oper. Res.* **1980**, *28*, 694–711. [CrossRef]
10. Sengupta, L.; Fränti, P. Comparison of eleven measures for estimating difficulty of open-loop TSP instances. *Appl. Comput. Intell.* **2021**, *1*, 1–30. [CrossRef]

11. Andrew, A.M. Another efficient algorithm for convex hulls in two dimensions. *Inf. Process. Lett.* **1979**, *9*, 216–219. [CrossRef]
12. Graham, R.L. An efficient algorith for determining the convex hull of a finite planar set. *Inf. Process. Lett.* **1972**, *1*, 132–133. [CrossRef]
13. Jarvis, R.A. On the identification of the convex hull of a finite set of points in the plane. *Inf. Process. Lett.* **1973**, *2*, 18–21. [CrossRef]
14. Anderson, K.R. A reevaluation of an efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.* **1978**, *7*, 53–55. [CrossRef]
15. Eddy, W.F. A new convex hull algorithm for planar sets. *ACM Trans. Math. Softw.* **1977**, *3*, 398–403. [CrossRef]
16. Bykat, A. Convex hull of a finite set of points in two dimensions. *Inf. Process. Lett.* **1978**, *7*, 296–298. [CrossRef]
17. Green, P.J.; Silverman, B.W. Constructing the convex hull of a set of points in the plane. *Comput. J.* **1979**, *22*, 262–266. [CrossRef]
18. Preparata, F.P.; Hong, S.J. Convex hulls of finite planar and spatial sets of points. In *Coordinated Science Laboratory Report no. UILU-ENG 75-2217, R-682*; University of Illinois at Urbana-Champaign, 1975. Available online: https://core.ac.uk/download/pdf/158320153.pdf (accessed on 19 August 2022).
19. Kirkpatrick, D.G.; Seidel, R. The ultimate planar convex hull algorithm? *SIAM J. Comput.* **1986**, *15*, 287–299. doi: 10.1137/0215021. [CrossRef]
20. Wenger, R. Randomized quickhull. *Algorithmica* **1997**, *17*, 322–329. [CrossRef]
21. Jünger, M.; Reinelt, G.; Rinaldi, G. The traveling salesman problem. In *Handbooks in Operations Research and Management Science*; Elsevier Science B.V.: Amsterdam, The Netherlands, 1995; Volume 7, pp. 225–330.
22. Bland, R.E.; Shallcross, D.F. Large traveling salesman problem arising from experiments in x-ray crystallography: A preliminary report on computation. *Oper. Res. Lett.* **1989**, *8*, 125–128. [CrossRef]
23. Ratliff, H.D.; Rosenthal, A.S. Order-picking in a rectangular warehouse: A solvable case of the traveling salesman problem. *Oper. Res.* **1983**, *31*, 507–521. [CrossRef]
24. Plante, R.D.; Lowe, T.J.; Chandrasekaran, R. The product matrix traveling salesman problem: An application and solution heuristic. *Oper. Res.* **1987**, *35*, 772–783. [CrossRef]
25. Lenstra, J.K.; Kan, A.R. Some simple applications of the traveling salesman problem. *J. Oper. Res. Soc.* **1975**, *26*, 717–733. [CrossRef]
26. Bektas, T. The multiple traveling salesman problem: An overview of formulations and solution procedures. *Omega* **2006**, *34*, 209–219. [CrossRef]
27. Cheikhrouhou, O.; Khoufi, I. A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy. *Comput. Sci. Rev.* **2021**, *40*, 100369. [CrossRef]
28. Samuel, G. Printing Press Scheduling for Multi-Edition Periodicals. *Manag. Sci.* **1970**, *16*, B-373–B-383. doi: 10.1287/mnsc.16.6.B373. [CrossRef]
29. Brumitt, B.L.; Stentz, A. Dynamic mission planning for multiple mobile robots. In Proceedings of the IEEE International Conference on Robotics and Automation, Minneapolis, MN, USA, 22–28 April 1996; Volume 3, pp. 2396–2401. [CrossRef]
30. Gilbert, K.C.; Hofstra, R.B. A New Multiperiod Multiple Traveling Salesman Problem with Heuristic and Application to a Scheduling Problem. *Decis. Sci.* **1992**, *23*, 250–259. [CrossRef]
31. Angel, R.D.; Caudle, W.L.; Noonan, R.; Whinston, A. Computer-assisted school bus scheduling. *Manag. Sci.* **1972**, *18*, B-279–B-288. [CrossRef]
32. Svestka, J.A.; Huckfeldt, V.E. Computational Experience with an M-Salesman Traveling Salesman Algorithm. *Manag. Sci.* **1973**, *19*, 790–799. [CrossRef]
33. Yu, Z.; Jinhai, L.; Guochang, G.; Rubo, Z.; Haiyan, Y. An implementation of evolutionary computation for path planning of cooperative mobile robots. In Proceedings of the 4th World Congress on Intelligent Control and Automation (Cat. No. 02EX527), Shanghai, China, 10–14 June 2002; Volume 3, pp. 1798–1802. [CrossRef]
34. Ryan, J.L.; Bailey, T.G.; Moore, J.T.; Carlton, W.B. Reactive tabu search in unmanned aerial reconnaissance simulations. In Proceedings of the 1998 Winter Simulation Conference (Cat. No. 98CH36274), Washington, DC, USA, 13–16 December 1998; Volume 1, pp. 873–879. [CrossRef]
35. Saleh, H.A.; Chelouah, R. The design of the global navigation satellite system surveying networks using genetic algorithms. *Eng. Appl. Artif. Intell.* **2004**, *17*, 111–122. [CrossRef]
36. Hosseinabadi, A.A.; Kardgar, M.; Shojafar, M.; Shamshirband, S.; Abraham, A. GELS-GA: Hybrid metaheuristic algorithm for solving Multiple Travelling Salesman Problem. In Proceedings of the 2014 14th International Conference on Intelligent Systems Design and Applications, Okinawa, Japan, 28–30 November 2014; pp. 76–81. [CrossRef]
37. Rostami, A.S.; Mohanna, F.; Keshavarz, H.; Hosseinabadi, A.A.R. Solving multiple traveling salesman problem using the gravitational emulation local search algorithm. *Appl. Math. Inf. Sci.* **2015**, *9*, 1–11. Available online: https://www.researchgate.net/publication/266392528_Solving_Multiple_Traveling_Salesman_Problem_using_the_Gravitational_Emulation_Local_Search_Algorithm (accessed on 13 July 2021).
38. Lo, K.M.; Yi, W.Y.; Wong, P.K.; Leung, K.S.; Leung, Y.; Mak, S.T. A genetic algorithm with new local operators for multiple traveling salesman problems. *Int. J. Comput. Intell. Syst.* **2018**, *11*, 692–705. [CrossRef]
39. Harrath, Y.; Salman, A.F.; Alqaddoumi, A.; Hasan, H.; Radhi, A. A novel hybrid approach for solving the multiple traveling salesmen problem. *Arab. J. Basic Appl. Sci.* **2019**, *26*, 103–112. [CrossRef]

40. Al-Furhud, M.A.; Ahmed, Z.H. Experimental Study of a Hybrid Genetic Algorithm for the Multiple Travelling Salesman Problem. *Math. Probl. Eng.* **2020**, *2020*, 3431420. [CrossRef]
41. Vakhania, N. Simple Methods for Euclidean Traveling Salesman Problems. In Proceedings of the 1st International Online Conference on Algorithms (IOCA 2021), Hangzhou, China, 22–24 November 2021. Available online: https://ioca2021.sciforum.net/#personnel1142 (accessed on 1 March 2022).
42. Pacheco-Valencia, V.; Vakhania, N.; Hernández, J.; Hernández-Gómez, J.C. A Fast Algorithm for Euclidean Bounded Single-Depot Multiple Traveling Salesman Problem. In Proceedings of the 1st Online Conference on Algorithms, Online, 27 September–10 October 2021; MDPI: Basel, Switzerland. Available online: https://sciforum.net/paper/view/10898 (accessed on 1 March 2022). [CrossRef]
43. Croes, G.A. A Method for Solving Traveling-Salesman Problems. *Oper. Res.* **1958**, *6*, 791–812. [CrossRef]
44. Universität Heidelberg, Institut für Informatik, Gerhard Reinelt, Symmetric Traveling Salesman Problem (TSP). Available online: https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ (accessed on 15 April 2021).
45. Natural Sciences and Engineering Research Council of Canada (NSERC) and Department of Combinatorics and Optimization at the University of Waterloo, TSP Test Data. Available online: http://www.math.uwaterloo.ca/tsp/data/index.html (accessed on 8 June 2019).
46. Google Maps. (n.d.) [Convenience Stores, OXXO, Located in Iguala, Guerrero, Mexico], Attribution: Map data ©2022 INEGI. https://goo.gl/maps/k5uteV2vFTcmvBaK8 (accessed on 11 July 2022).
47. Antia Viajes. Available online: https://antiaviajes.com/asia/viajes-a-china/corea-japon (accessed on 11 July 2022).
48. Top Asia Tour. China International Travel Service Guillin Co., Ltd. Available online: https://es.topasiatour.com/viajes/tates-corea-del-sur-04.htm (accessed on 11 July 2022).
49. Google Maps. (n.d.) [Tourist Places in Asia]. Attribution: Map Data ©2022 Google, TMap Mobility. https://goo.gl/maps/opy3QELuuwZp4g9z6 (accessed on 11 July 2022).
50. Viajes, Visas y Excursiones–Vive. Available online: https://www.viajesvivex.com/circuitos (accessed on 11 July 2022).
51. Google Maps. (n.d.) [Tourist Places in Europe], Attribution: Map Data ©2022 Google, GeoBasis-DE/BKG(©2009), Inst. Geogr. Nacional. https://goo.gl/maps/8AjGcNcsGRyDQgCR8 (accessed on 11 July 2022).
52. AndesNomads.com Servicios Turísticos. Available online: https://www.travelandes.com/ (accessed on 11 July 2022).
53. Google Maps. (n.d.) [Tourist Places in South-America], Attribution: Map Data ©2022 Google. https://goo.gl/maps/uk35NwKRYwmo37FA8 (accessed on 11 July 2022).
54. Tours-USA.com. Available online: https://www.tours-usa.com/tour/the-grand-east (accessed on 11 July 2022).
55. Google Maps. (n.d.) [Tourist Places in East of U.S.A], Attribution: Map Data ©2022 Google, INEGI. https://goo.gl/maps/W62uQcgrSzbUp8ni8 (accessed on 11 July 2022).
56. LyO Gestión y Ocio, S.L. Available online: https://www.viajeslyo/ (accessed on 11 July 2022).
57. Evaneos.es. Available online: https://www.evaneos.es/ (accessed on 11 July 2022).
58. Google Maps. (n.d.) [Tourist places in Mexico], Attribution: Map data ©2022 INEGI. Available online: https://goo.gl/maps/79xJXgQTzCitQbRz5 (accessed on 11 July 2022).