University of Molise

Bioscienze e Territorio XXXIV cycle

Curriculum

Informatico-Matematico

The Close Enough Traveling Salesman Problem: enhanced heuristics, applications and variants

Tutor

Prof. Carmine Cerrone

Candidate

Andrea Di Placido

PhD Coordinator

Prof. Giovanni Fabbrocino

Contents

Outline

The traveling salesman problem (TSP) is widely studied in the literature for its ability to model a wide range of problems. The goal is to identify the minimum path that visits each city (target, customer, etc.) of a given set precisely once and returns to the starting city, commonly referred to as the depot. The TSP has several practical applications concerning planning, logistics, microchip manufacturing, and even DNA sequencing. In recent years, generalizations of the TSP are highly treated for logistics purposes that are the close enough traveling salesman problem (CETSP) and the close enough arc routing problem (CEARP). In the CETSP, each target has a range of action that is considered visited if traversed. Then, the salesman is not bound to reach the customer location but must get close enough to each customer to visit it. We are not constrained to a road network in this problem, so the distance between points is Euclidean. Instead, in the case of CEARP, we follow the same logic as CETSP, but we are constrained to follow a set of arcs (e.g., road network) to define our tour. The CETSP and CEARP have practical applications in several real-world scenarios. For instance, suppose we need to perform meter readings in a neighborhood. Typically, this is done by arriving on-site and performing the reading. If the meters are equipped with systems that allow remote reading, such as radio-frequency identification systems (RFID), the reading can be done by simply crossing the RFID range. An RFID system consists of a radio transponder that transmits data about its tag when interrogated by an electromagnetic pulse from another RFID system. We develop a genetic algorithm to solve CETSP. In this algorithm, we combine several techniques to provide excellent solutions. The algorithm exploits 2opt to improve the route length and solves a conic programming problem to optimize the position of points given a fixed sequence. The conic model is solved sporadically and replaced by a bisection algorithm that can provide results comparable to the model. More explanation of the procedures described above can be found in the section 2.3.7. Also, we provide the concept of *turning point*, which is a point related to a specific target, s.t. the distance between the point and the target is within the range. We compare the results to those produced by three methods presented in the literature. Our approach provides the best output in 59 out of 62 cases.

Moreover, we formulate two metrics for evaluating a CETSP, TSPDegree (TSPD) and OverlappingCenter (OC). TSPD uses the distances between targets and radii to estimate the morphology of an instance, while OC assesses its difficulty through the conformation of the solution. We compared our metrics to the overlap ratio (OR), a measure already in the literature presented by Mennell. We evaluated a set of instances to see if common patterns identified difficult instances. By challenging, we mean a set of computationally burdensome instances to solve in terms of time. The results show that TSPD and OC provide more information about the characteristics of instances than OR. Furthermore, we found that difficult instances correspond to average values of TSPD and OC.

Finally, we presented a case study related to solar panel diagnostics. Specifically, a drone equipped with a thermal camera was used to check the correct operation of a photovoltaic field. The thermography of a panel gives information on the operating temperature of the entire panel and the individual cells which compose it. The following conditions must be met to ensure that a thermal photo can be used for diagnostics.

- The photo must be taken in such a way that the panel is framed correctly (see Figure 2.16, page 54), i.e., the flight level and the position in which the photo is taken with respect to the panel inclination should be such that the panel is correctly detected.
- Photos should be taken as quickly as possible to ensure their correct use during diagnosis.
- The time available for determining the drone's route is limited.

Given the application scenario just described, it was necessary to design an algorithm to solve the $CETSP$ by producing high-quality solutions in short computation times (a few minutes). Before developing the GA algorithm proposed in this dissertation, the drone's route was determined as the TSP route over the centers of the circles. The savings obtained by applying the GA amounted to 15% of the route length.

As part of this project, it was necessary to develop an approach capable of producing convolution filters to identify artifacts or patterns present in an image. Comparing the thermographs of the panels connected to the same inverter is crucial to verify the correct functioning of the individual panels. The comparison is necessary to understand if there is an overuse of some panels connected to the same String or some Strings compared to other Strings with the same energy absorption by the inverter. Moreover, the image analysis in question could be disrupted by anything from scratch or erosion of the panel itself. One of the basic techniques used for image processing is convolution and its inverse, deconvolution. Convolution is a mathematical operation that, given two functions f and g as input, produces a third that expresses how the first one has been modified by the other. In contrast, deconvolution is an algorithmic process used to reverse convolution effects on data. Specifically, it aims to identify the function h such that $f \times g = h$. While it is always possible to compute convolution, it is not always possible to compute deconvolution due to loss of information. Therefore, it was necessary to define an approach capable of extracting convolution filters from the original and filtered images. We defined a mathematical formulation that extracts filters equal to or equivalent to the filter applied on the original image. The importance of this approach concerns the possibility of modeling the problem exactly and allows us to certify the goodness of the obtained filter. In this way, it is possible to certify that a given transformation results from a convolution. The results show that our approach is competitive with the state of the art when the images are free of noise.

The mixed constrained generalized routing problem (MCGRP) generalizes both CETSP and CEARP. Considering the CETSP as closely related to drones, it refers to a drone flying freely in space. Unfortunately, this scenario is not always true. If we consider locations such as schools, hospitals, military or residential areas, flying over these areas is usually prohibited for several reasons such as safety, public order, privacy, etc. In the MCGRP, we introduce the concept of a flight zone. We differentiate two: one in which the drone can fly freely, called free-flight zone (FFZ), and a second one in which the possibility of flight is limited to specific corridors (e.g., roads) or prohibited, called constrained flight zone (CFZ). Given a graph $G = (V, A, M, Z)$, where V is the set of targets to serve, A is the set of arcs (e.g., roads), M is the set of nodes (e.g., crossing between streets), and Z is the set of CFZ, we need to find the route that starts and ends at the depot, and minimize the total sum of edge lengths. We have formally defined the problem and examined it in its CEARP and especially CETSP edge cases. Future developments include defining approaches that can also handle intermediate situations, with variable mixes of CFZs and FFZs, and examining its behavior concerning its boundary cases.

The generalized close enough traveling salesman problem (GCETSP) is a variant of the CETSP in which a set of areas is associated with each target. We identified these areas as concentric disks with variable radius in the specific case considered. Each of these disks is associated with a prize. Formally, let C be a set of customers in a Euclidean plane, and let c_0 be the first customer, named *depot*. For each customer $c \in C \backslash c_0$, a set of K disks are defined, all centered at c and with increasing radius, i.e., $r_c^k < r_c^{k+1}, k = 1, ..., |K| - 1$, where r_c^k is the radius of the disk k centered at c. In addition, a *prize* p_c^k is associated with each disk k, with $p_c^k > p_c^{k+1}, k + 1, ..., |K| - 1$. The depot is associated with one disk of radius 0 and prize 0. The GCETSP aims to determine the route that maximizes the difference between the total collected prize and the route length and visits one disk per customer. The GCETSP can model several application scenarios where major benefits occur through attaining proximity to the targets. For example, in RFID meter reading systems, prizes may represent the probability of a successful meter reading at a customer's home (the target in this case). This probability decreases as the distance from the customer increases, which explains why the prize decreases with respect to the disk's radius. We reconsidered the instances of CETSP found in the literature for this new variant and defined a new set of instances for GCETSP. We formulated constructive heuristics for resolution and adapted the previously developed genetic algorithm for CETSP to this new variant. The results show that the GA can solve the problem correctly, defining excellent solutions. Comparing the two approaches, GA produces the best solutions in most cases, while the heuristic approach still provides reasonable solutions comparable with those of the genetic.

This dissertation will report in detail on all of the above. Following this outline, we will thoroughly review the literature and state of the art. In chapter 2 we report everything about CETSP, while in chapter 3 and 4 we show in detail the variants we proposed for this problem. Finally, in chapter 5 we will discuss the conclusions and future developments of this thesis.

Chapter 1

Literature review

Over the years, technological advancement has motivated new problems in operations and logistics research. CETSP and CEARP are some of these. The classic TSP requires a salesman to visit customers at their precise location. For example, utility companies send their workers to read meters in residential neighborhoods. However, it is possible to read meters remotely with radio frequency identification technology. Nowadays, instead of visiting each customer on-site, we need to be close enough to serve them. Similarly, if many customers are evenly distributed on the street, crossing that street corresponds to serving those customers.

Generally, this problem is defined on a Euclidean plane. The salesman must start and complete his route to the depot. Each customer has a region where he can be served and is considered visited when the salesman passes through or touches the region. The goal is to visit all customers and return to the depot by traveling the minimum distance. The customer action area is assumed to be circular (a disk), with a specific radius greater than zero, and with a center at the customer's location. If all radii are zero, then the CETSP reduces to a TSP and the CEARP to a Rural Postman Problem (RPP). In the general case, where all customer radii are positive, the location where the salesman will visit the customer is not known a priori. Therefore, it is necessary to define the sequence of customer visits and in which position those customers will be served. In CETSP, we can place the points on the circumference surrounding the customer, while in CEARP, the intersections between the arcs and circumferences determine the position of the points.

[Gulczynski et al., 2006] were among the first to examine a common case of CETSP where the areas around all nodes are disks of the same radius. This problem describes situations where RFID tags are used to deliver data for collection remotely. The use of RFIDs allows utility companies to read meters remotely. Then an operator can take the reading without visiting the customer but pass within a certain radius of each customer. This concept allows us to translate the problem from a TSP to a CETSP. The authors investigated six different heuristics produced to solve the problem. All six heuristics follow three general steps: given an initial set of customers C, the first step focuses on creating a set of eligible supernodes S. An admissible set of supernodes is a set of points (also containing the depot) with the property that for each customer node, a point in the set is within its range. In Figure 1.1, the set of asterisks represents an admissible supernode set since for each customer (circle), there is an asterisk in the radius of r units ($r = 9$). After producing S, the second step is to define a near-optimal TSP tour, T, on the points of the set S. For the definition of supernode set, we can guarantee that the tour T traverses all customers in their neighborhoods. Thus, T is an eligible tour for CETSP. Since the cardinality of the supernode set is smaller than the number of clients, sometimes meaningfully, it turns out to be more efficient to generate the tour on S. Thus, running step 1 before step 2 takes significantly less computational time than starting to generate the TSP tour from C. The last step is an economizing routine that reduces the traveled length of T while maintaining its eligibility. The results show that these heuristics produce CETSP tours that are significantly shorter in length than the TSP tour for customers.

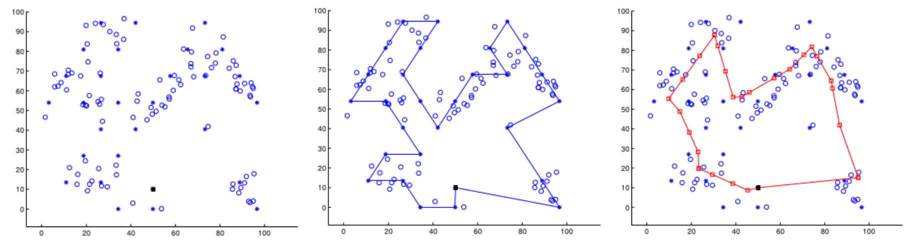


Figure 1.1: An example CETSP tour on 100 nodes, with radius 9, and the depot located at (50,10). The circles represent the customer nodes, and the asterisks are the supernodes. The red path represents the results of an economization of the tour. [Gulczynski et al., 2006]

[Shuttleworth et al., 2008] also proposed other heuristics to solve the automatic meter reading shortest tour problem (AMRSTP) on a road network. The solutions generated by the proposed heuristics were compared to those of the rural postman problem solvers and showed improvements in solution values. In addition, [Dong et al., 2007] also introduced two heuristics, based on the concept of supernodes, to approach the AMRSTP problem. The supernodes are generated using convex hull and clustering techniques. Moreover, a mixed-integer nonlinear programming formulation is provided for CETSP, but it is not explicitly used in the algorithm design since the exact resolution is impractical. The algorithms were tested on various datasets, and the results report that both heuristic algorithms perform effectively and efficiently.

[Yuan et al., 2007] presented a new algorithm for solving the traveling salesman problem with neighborhoods (TSPN). Specifically, the authors report a case of robot routing problems in wireless sensor networks. The innovation proposed by the authors with the approach is to build the TSP tour from the beginning, starting from the initial position and disk centers. Then, advanced optimization techniques are built to improve the produced route. The validity of this heuristic was evaluated by a direct relationship between the TSPN and the TSP. For instance, when the size of the disks is equal to zero, the TSPN is identical to the TSP. Despite very high computational times, the methodology can identify the shortest tour in all benchmark instances, providing significant improvements over the last approximation algorithm for the problem. A visual comparison between a TSP tour and a TSPN tour is shown in fig. 1.2.

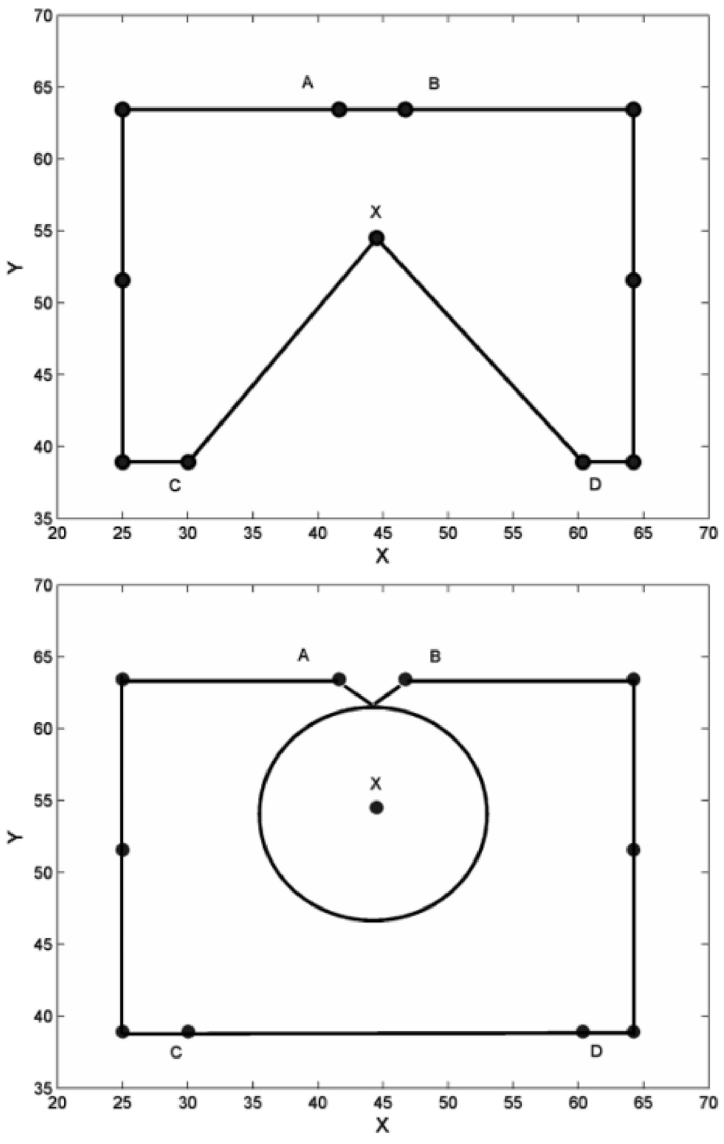


Figure 1.2: An example of TSP optimal tour vs TSPN optimal tour [Yuan et al., 2007]

[Arkin and Hassin, 1994], [Mata and Mitchell, 1995], and [Dumitrescu and Mitchell, 2003] described some approximation algorithms that can solve special cases of CETSP in polynomial time.

[Arkin and Hassin, 1994] provided an approximation algorithm that uses symbolic points for each neighborhood. The algorithm was implemented for a particular case in which all neighborhoods have diameter segments parallel to a common direction, and the ratio between the largest and smallest diameter is fixed by a constant. [Mata and Mitchell, 1995], on the other hand, produced approximation algorithms in the case where the neighborhoods had an arbitrary polygonal shape.

[Dumitrescu and Mitchell, 2003] obtained several results on the TSPN geometric: first, they extended the approaches defined in [Arkin and Hassin, 1994] and obtained the first O(1)-approximation algorithm for a particular case of TSPN, in which there are connected regions with the same diameter. Next, they provided a polynomial-time approximation scheme in the case of disjoint disks. Finally, they obtained improvements on the previous approximation of [Arkin and Hassin, 1994].

[Mennell, 2009] has recently produced a heuristic that has been tested on large-size problems. Based on the 3-steps scheme, the author implemented this heuristic using the concept of Steiner-zones, non-empty intersections between neighborhoods. Based on this, first, the algorithm identifies a set of Steiner-zones that includes all the neighborhoods; then, it discretizes the Steiner-zones into a collection of points and computes the TSP tour on them. Finally, the tour is enhanced by improving the position of the discretization points. The results show that the approach can identify good quality solutions reasonably. In addition, the author provided a new set of instances for CETSP: 48 fixed-radius and 14 variable-radius. For more details on them, refer to sec. 2.5. Finally, [Mennell et al., 2011] provided a second-order cone programming model (SOCP) to solve the touring Steiner zones problem (TSZP) when the tour sequence is given and a first mathematical formulation for the CETSP.

[Behdani and Smith, 2014] have tried to solve CETSP at the optimal. The authors proved two fundamental properties of optimal solutions to the problem: the first exposes how all optimal solutions are composed of a finite number of connected segments, where at least one point lies on the perimeter of a target neighborhood. The second proves that if a tour T is an optimal tour, then the points that make up the tour lie within the convex hull. These assumptions allowed the authors to devise a discretization scheme. They produced valid inequalities and attempted to solve the problem at the optimum via three procedures:

1. a mixed-integer program formulation;
2. benders decomposition,
3. an iterative algorithm.

They produced 240 new test instances with different cardinalities, from 7 to 30 nodes. The authors tested their procedures on these instances, and the results show that no instance was solved to the optimum. Nevertheless, reasonable upper and lower bounds were obtained for these instances.

[Carrabs et al., 2017b] introduced a novel discretization model that improves the results of [Behdani and Smith, 2014]. They showed that a better neighborhood approximation can be obtained by considering interior instead of exterior points. For example, a fairly intuitive discretization scheme consists of placing points associated with targets on the perimeter. This scheme is called the **perimeter discretization** (PD) scheme. In detail, let k be the number of points used to discretize each neighborhood $N(v)$. Then the PD scheme divides each circumference C_v into k equal arcs, and places the discretization points at the ends of these arcs. Let α be the angle associated with each arc $\widehat{d_1, d_j}$, then $\alpha = \frac{2\pi}{k}$. A PD scheme with $k = 3$ is shown in Fig. 1.3, resulting in $\alpha = 120$ and $N(v) = \{d_1, d_2, d_3\}$. Evaluating the maximum discretization error associated with a PD scheme, the worst case happens when the point p_i of the tour $T*$ touches the circumference C_v in the middle of the arc $\widehat{d_1, d_j}$. In fact, in this case the distance between p_1 and the nearest discretization point d_1 (or d_2) is maximal. Hence, $w(\overline{p_1, d_2}) = 2r_v sin(\frac{\alpha}{4})$ and then $\xi(v) = 4r_v sin(\frac{\alpha}{4}))$. In general, given the number of discretization points k, the maximum error for a PD scheme is $\xi(v) = 4r_v sin(\frac{\pi}{2k})$.

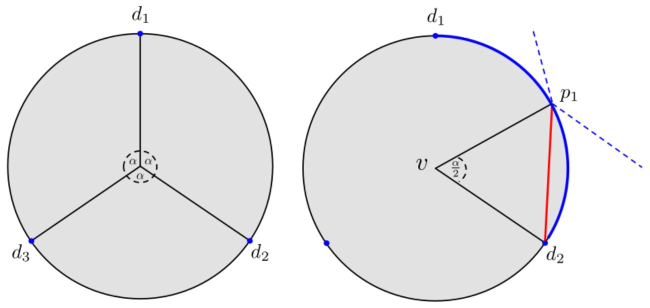


Figure 1.3: Perimetral discretization for $k = 3$, with $N(v) = \{d_1, d_2, d_3\}$ and $\alpha = 120$. The scheme is shown on the left, while the maximum discretization error is reported on the right.[Carrabs et al., 2017b, Carrabs et al., 2017a].

The new discretization scheme proposed by the authors is called **internal point discretization** (IP). Given the number of discretization points k, the IP scheme divides the circumference C_v into k circular arcs and, for each arc $\widehat{a, b}$ places a discretization point in the middle of the chord $\overline{a, b}$. An example is shown in Fig. 1.4. Applying the IP scheme ensures that when a contact point p_1 of the tour $T*$ lies on the arc $\widehat{a, b}$, the maximum distance between p_1 and the nearest discretization point is equal to half the length of the segment $w(\overline{a, b})$. Consequently, for a fixed number k, the maximum error associated with an IP scheme is equal to $\xi(v) = 2r_v sin(\frac{\pi}{k})$.

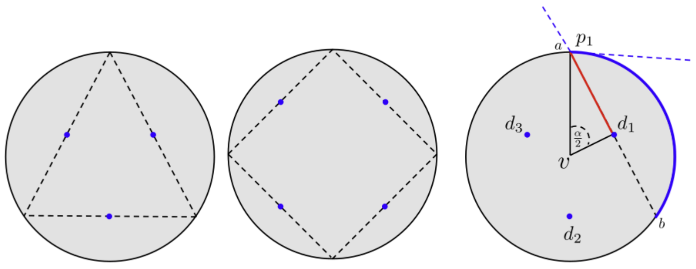


Figure 1.4: Internal discretization for $k = 3$ and $k = 4$. The scheme are shown on the left, while the maximum discretization error is reported on the right. [Carrabs et al., 2017b, Carrabs et al., 2017a].

The authors provided improved upper and lower bounds for instances of [Behdani and Smith, 2014] in a reasonable time.

[Coutinho et al., 2016] proposed an exact branch-and-bound (B&B) algorithm that can provide optimal solutions for instances of [Behdani and Smith, 2014] and [Mennell, 2009]. The method works as follows: each B&B node is associated with a partial optimal tour that needs to visit only a portion of vertices in a particular order. The algorithm chooses three vertices to generate an initial sequence: the first vertex to be chosen is necessarily the depot. The next vertex to be chosen is the one furthest from the depot. The third to be inserted leads to the largest lower bound value. Specifically, the method solves a SOCP for all remaining candidates and selects the vertex associated with the sequence to the best relaxation. An example with seven vertices is illustrated in Fig. 1.5. We can observe how the depot (vertex 0) is the first to be chosen for the sequence, followed by vertex 6 (the farthest from the depot) and vertex 3, following the previously mentioned insertion criterion.

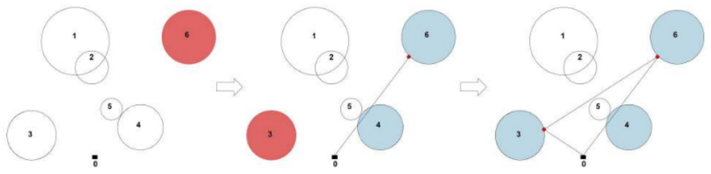


Figure 1.5: Example of the procedure to find a valid root relaxation. [Coutinho et al., 2016].

Since only three vertices are involved in this sequence and the symmetric costs, their order does not affect the solution. Therefore, this partial tour is a good relaxation for the main problem. Choosing the exact coordinates of the tour points to be visited can be formulated as a SOCP. Then, if the associated solution is admissible, i.e., all the customers are visited, the obtained solution is optimal, and the problem is solved. Otherwise, the algorithm defines three more branches of subproblems where, in each of them, a vertex that is not present in the sequence is inserted in a different position. A node in the B&B is pruned if its cost is greater than or equal to the best upper bound known at the time or if the associated solution is admissible. Otherwise, the algorithm creates branches using the same reasoning applied for the root node. The presented approach can solve at the optimum all instances of [Behdani and Smith, 2014]. Relative to the instances of [Mennell, 2009], it solved several instances at the optimum, including large ones, and improved the known lower bounds of the remaining ones.

More recent techniques for solving CETSP have been proposed by [Yang et al., 2018], [Wang et al., 2019] and [Carrabs et al., 2020].

[Yang et al., 2018] presented a hybrid algorithm (HA) that combines a genetic algorithm (GA) with particle swarm optimization (PSO) to solve CETSP with arbitrarily shaped neighborhoods (it trivially reduces to a CETSP if the neighborhoods are disks). The PSO algorithm has been presented by [Kennedy and Eberhart, 1995] as a metaheuristic inspired by the behavior of flocks of birds and fish, capable of exploring large portions of the space of candidate solutions. GA was introduced by [Holland et al., 1992] and is a global optimization algorithm inspired by biological evolutionary processes such as inheritance, natural selection, genetic crossing over, and mutation. HA is a hybrid collaboration-based optimizer for which hybridization occurs at the component level of the solution, and two-parent optimizers operate in different component spaces. PSO is used to generate a point set and iteratively improve it, while GA identifies the TSP sequence and minimizes it in terms of distance traveled. The combination of the two techniques ensures high-quality solutions in short computational times. HA has been tested on instances of [Mennell, 2009] and the results are comparable with other approaches in the literature.

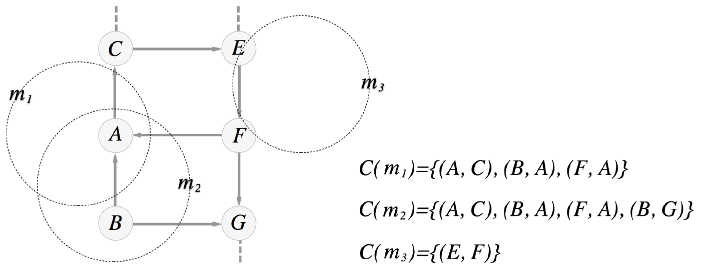
[Wang et al., 2019] have recently proposed a new approach based on Steiner-zones, referred to as Steiner zone variable neighborhood search (SZVNS). The algorithm works in three steps. The first one is a phase of reducing the targets to be covered: it is possible to reduce the problem size according to some observations. For example, it is possible to eliminate a target C_i if its disk contains that of another target C_j; any tour that visits C_j will surely visit C_i as well. However, if all targets have disks with the same radius, no target can be eliminated unless they coincide. Similarly, if a disk contains the depot inside it, it is automatically visited. The second step concerns finding the Steiner zones and constructing an admissible tour that traverses all of them. Finally, the third phase is all about improving the solution iteratively. The approach has been tested on instances of [Mennell, 2009], and the results show that SZVNS produces better and comparable results in less time than state-of-the-art algorithms.

[Carrabs et al., 2020] have recently proposed a new iterative procedure, called $(lb/ub)Alg$, which can further improve the results obtained so far. The algorithm calculates the lower and upper bound at each iteration using the *lbc* (lower bound procedure) and *ubc* (upper bound procedure). These functions are influenced by parameters that are updated at each iteration. The *lbc* procedure selects a subset of targets and discretizes their neighborhoods. Then, it defines a tour by solving a generalized traveling salesman problem (GTSP) on the graph induced by the discretization points. Finally, the lower bound is computed by subtracting the errors of each discretization on each node. The focal point of *lbc* is the choice of subset, which is performed through a greedy carousel [Cerrone et al., 2017a]. The greedy carousel is an enhanced greedy algorithm for which its effectiveness and computational efficiency have been demonstrated on a wide variety of subset selection problems. The selection of the most promising target points is performed by the *FindNextPoint* procedure: this assigns to each unselected target v a value $val(v)$ based on the distance from the previously defined subset points. More considerable distances correspond to a higher value. Then, a subset of vertices with the lowest values is chosen. The weighted average μ is computed, and, finally, the score $val(v)$ is computed by multiplying μ with two parameters Δ and Γ_v and subtracting the result with the discretization error. The most promising target is the one with the maximum score. $(lb/ub)Alg$ focuses on these parameters since they condition the construction of the tour and the lower bound quality produced by *lbc*. Also, providing several starting tours to *ubc* is more likely to improve the final upper bound. $(lb/ub)Alg$ obtains good results on the majority of instances of [Mennell, 2009] by providing new upper and lower bounds for these instances.

Regarding CEARP, the first to introduce the problem were **[Drexl, 2014]**. The authors considered many practical applications of single-vehicle routing problems that can be modeled with a generalization of the rural postman problem, the generalized directed rural postman problem (GDRPP). Specifically, they considered cases such as problems with real constraints, such as turn penalties, street segment side, zigzag service, use of public transport, etc.; hierarchical postman problem, and generalized traveling salesman problem. Among these cases, it has also been considered the arc routing problem modeled on a directed graph, duplicating the arcs of the graph for both travel directions. The authors demonstrated the versatility of the GDRPP as a unique model for postman problems. In addition, they presented a solving heuristic and an exact model for its resolution and tested it on two large sets of benchmark instances.

[Ha et al., 2012] have studied a variant of CETSP in which there are constraints on the coverage of the arches. This problem was introduced in utility companies that use RFID-enabled automatic meter reading technologies. The authors propose a mathematical formulation and an exact algorithm for solving the problem. The results show that the exact approach can solve realistic size instances with 1000 arcs and 9000 customers in less than two hours. Next, [Ha et al., 2014] considered CEARP and its interesting practical applications. In [Ha et al., 2014], the authors provide a new mathematical formulation for the problem, comparing it with those found in the literature. They also present a branch-and-cut algorithm for the optimal resolution of the problem.

[Cerrone et al., 2017b] proposed a novel graph reduction technique and flow formulation for CEARP. The graph reduction technique is based on the fact that only a subset of arcs are needed and, therefore, many others are redundant. An example is shown in Figure 1.6.



C
E
m₁
m₃
A
F
m₂
B
G
C(m₁)={(A, C), (B, A), (F, A)}
C(m₂)={(A, C), (B, A), (F, A), (B, G)}
C(m₃)={(E, F)}

Figure 1.6: Example of the redundant arches procedure proposed in [Cerrone et al., 2017b].

Next, a vertex cover problem is solved to identify which nodes are needed to obtain an acceptable solution. Based on this preprocessing, the authors defined a new MIP formulation for the problem, which we report below. Given a directed graph $G = (V, A, M)$, where V is the set of nodes, A is the set of arcs, and M is the set of targets positioned along the arcs. Let N be the set of pairs (m, a) with $m \in M$ and $a \in A$ such that a traverses the target m. Let c_{ij} be the cost associated with the arc (i, j). For each arc $(i, j) \in A$, let x_{ij} be a variable indicating the number of times the arc is crossed, and f_{ij} be the associated flow variable. We want to optimize the objective function:

$$Minimize \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1.1}$$

s.t.

$$\sum_{a=(i,j)\in A|(m,a)\in N} x_{ij} \geq 1 \qquad \forall m \in M \qquad (1.2)$$

$$x_{ij} \geq 1 \qquad \forall (i,j) \in \widehat{A} \qquad (1.3)$$

$$\sum_{j\in V|(i,j)\in A} x_{ij} - \sum_{j\in V|(j,i)\in A} x_{ji} = 0 \qquad \forall i \in V \qquad (1.4)$$

$$\sum_{(0,j)\in A} f_{0j} \geq 1 \qquad (1.5)$$

$$\sum_{j\in V|(i,j)\in A} f_{ij} - \sum_{j\in V|(j,i)\in A} f_{ji} = 0 \qquad \forall i \in V \qquad (1.6)$$

$$(VC \bigcup \{0\}) \qquad (1.7)$$

$$\sum_{j\in V|(i,j)\in A} f_{ij} - \sum_{j\in V|(j,i)\in A} f_{ji} = 1 \qquad \forall i \in \widehat{V\{0\}}) \qquad (1.8)$$

$$\sum_{j\in V|(i,j)\in A} f_{ij} - \sum_{j\in V|(j,i)\in A} f_{ji} = \sum_{j\in V|(i,j)\in A} x_{ij} \qquad \forall i \in VC\backslash(\widehat{V} \bigcup \{0\}) \qquad (1.9)$$

$$f_{ij} \leq M x_{ij} \qquad \forall (i,j) \in A \qquad (1.10)$$

$$x_{ij} \in Z_0^+ \qquad \forall (i,j) \in A \qquad (1.11)$$

$$f_{ij} \in R_0^+ \qquad \forall (i,j) \in A \qquad (1.12)$$

The constraints (2.2) ensure that every target is covered by at least one arc in the solution, and the constraints (2.3) ensure that every necessary arc is present in the solution. The family of constraints (2.4) ensures that the number of outgoing arcs selected for each node is equal to the number of incoming ones. Constraint (2.5) sets the outgoing flow from the depot equal to zero. Constraint set (2.6) prevents flow from passing through all unneeded vertices, asserting the connectivity of the final solution, while constraints (2.7) force the presence of flow at all vertices necessary for the admissible solution. Constraints (2.8) set that for every vertex that could be used to ensure connectivity of the solution, the incoming flow is equal to the time the vertex is traversed in the solution. Finally, the set of constraints (2.9) ensures that flow is only present in the arcs used in the solution. The authors performed several experiments on oriented graphs and showed the effectiveness of the graph reduction technique. In addition, the proposed mathematical formulation was compared with other exact methods in the literature, showing that their approach is efficient in practice.

Chapter 2

Close Enough Traveling Salesman Problem

2.1 Introduction

The close enough traveling salesman problem (CETSP) is a generalization of the classical traveling salesman problem (TSP) problem. A set of nodes (customers, targets) is defined, and each node is associated with an area, called a neighborhood, surrounding it. Generally, this area is circular. The goal of CETSP is to determine the shortest tour that starts at a point, defined as *depot*, traverses the neighborhood of each target once, and returns to the depot. CETSP models several real-world applications: e.g., meter reading using RFID systems [Shuttleworth et al., 2008], forest fire localization using aerial vehicles [Poikonen et al., 2017], robot monitoring of wireless sensor networks [Yuan et al., 2007]. This dissertation also introduces a novel application regarding solar panel diagnostics via aerial vehicles, such as drones. While CETSP has been studied in the past under several names, such as the covering salesman problem [Current and Schilling, 1989], the geometric covering salesman problem [Arkin and Hassin, 1994], the covering tour problem [Gendreau et al., 1997], the Euclidean TSP with neighborhood [Dumitrescu and Mitchell, 2003], the problem was first named CETSP by [Gulczynski et al., 2006]. Several algorithms have been proposed in the literature to provide reasonable upper and lower bounds regarding the solving methods for the problem. For an in-depth discussion on this, refer to the previous chapter. The main contributions made to the literature are as follows:

- We propose a genetic algorithm (GA) for solving CETSP. As reported in [Wang et al., 2019], there is room for improvement on the state-of-the-art solving heuristics. GA provides solutions comparable to or better than those produced by the approaches in the literature in relatively little time.

- We propose metrics based on the characteristics of the problem instances to identify which has the greatest impact on the difficulty of solving the problem. We also revise the metric already proposed by [Mennell, 2009], overlap ratio, and correct its calculation made in previous works.

- We present a case study related to reconnaissance for solar panel diagnostics. The case study is related to a research project developed at the University of Molise. We show how the problem under study is modeled perfectly through a close enough traveling salesman problem, and therefore solvable with GA, focusing on the benefits of this solving approach.

These results have been compiled into [Di Placido et al., 2021] and submitted to the journal Computers and Operations Research. The work has been accepted and is under review. The chapter is structured as follows. The problem and the notation used are described in section 2.2. Section 2.3 is devoted to the solving approach, while in section 2.4 we present the proposed new metrics for classifying the instances of the problem and also present the corrected overlap ratio calculations. The computational results and the case study are given in the 2.5 and 2.5.3 sections, respectively. Finally, in the **??** section, we give a brief abstract of the work [Capobianco et al., 2021], as it is an integral part of the above project.

2.2 Problem Definition

We report below the definition and notations found in the work [Di Placido et al., 2021].

"Let N be a set of nodes in a plane, and let p_0 be starting node named *depot*. We refer to the elements of N as targets. Each target $v_i \in N$ is associated with a neighborhood $N(v_i)$ corresponding to a circle (disk) with outer circumference C_{v_i} centered in v_i and with radius r_{v_i}. The goal of the close enough traveling salesman problem(CETSP) is to find the shortest tour $T = \{p_0, p_i, ..., p_k, p_0\}$, $k = |N|$, that starts from the depot p_0, traverses every neighborhood $N(v_i)$, and ends in p_0. Points p_i, $i \neq 0$, visited in tour T are called *turning points*. Each turning point p_i is associated with a target v_i, i.e., $d(p_i, v_i) \leq r_i$, where $d(p_i, v_i)$ is the Euclidean distance between p_i and v_i. This means that a turning point p_i is the point in which the tour intersects the neighborhood of target v_i. For target $v_i \in N$, the turning point p_i could be unique or any point of the segment which traverse $N(v_i)$. An illustrative example is provided in Fig. 2.1

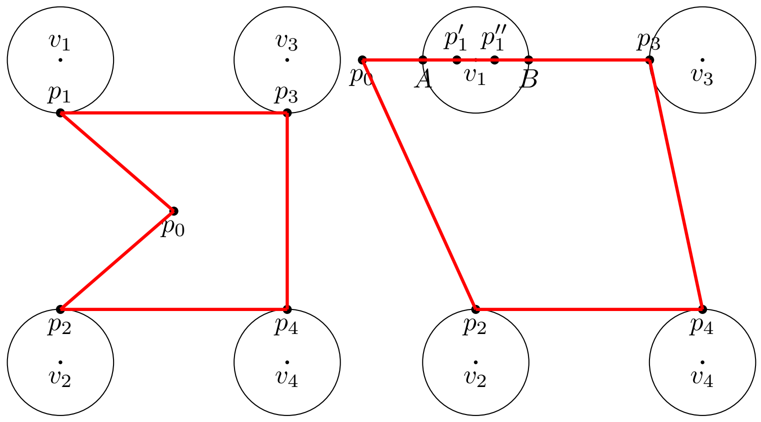


Figure 2.1: Examples of CETSP solutions. Each target v_i is associated with a neighborhood $N(v_i)$ and a turning point p_i. The red lines represents the tour T that starts from the depot p_0, touches neighborhoods $N(v_i)$ in the turning points p_i, and ends in p_0. On the left, the turning point p_1 is unique, while on the right it could be any point on the segment $\overline{AB}$. [Di Placido et al., 2021]

Given a pair of turning points p_i and p_j, the distance between them is given by the Euclidean distance $d(p_i, p_j)$. Thus, the total cost of a tour T is equal to $\sum_{n=1}^{|N|-1} d(p_i, p_{i+1})$.

We say that a tour T covers a target v_i if T intersects $N(v_i)$ in at least one point. The goal of the CETSP is to determine the shortest tour T covering all targets."

2.3 Genetic Algorithm

Genetic algorithms (GAs) are evolutionary algorithms introduced by [Holland et al., 1992] that have been widely used to solve combinatorial optimization problems. GA applies the Darwinian concept of natural selection on a set of admissible solutions. Each solution is composed of *genes* that evolve over generations through techniques inspired by biological processes such as selection, mutation, and [Mitchell, 1998] crossover. Through GA, it is possible to explore a large portion of the solution space by modeling different admissible solutions (population) and carefully choosing the behavior of the biological operators.

In the following, we describe how we devised and designed a GA to solve CETSP.

2.3.1 Preprocessing Procedures

We perform two preprocessing procedures to reduce the magnitude of the problem by removing targets that are automatically visited when covering other ones, as reported in the following observations:

O1: *Consider a target $v_i \in N$ for which $p_0 \in C_{v_i}$. Then, as any feasible CETSP solution visits p_0, v_i is included in any feasible solution and can be removed.*

O1 is provable by hypothesis: given the definition of CETSP, p_0 is the depot from which all eligible tours begin and end. If the depot is contained in the neighborhood of a target $N(v_i)$, it means that the target v_i is automatically visited, since all tours will pass through p_0, hence through $N(v_i)$. To be clear, p_0 is contained in $N(v_i)$ if $d(p_0, v_i) \leq r_{v_i}$

O2: *Let $v_1, v_2 \in N$ be two targets. If $N(v_2) \subset N(v_1)$ then v_1 is covered by all the feasible tours passing through $N(v_2)$ and can thus be removed.*

O2 is easily deduced: an eligible tour T must cover all targets contained in N. Since all points of $N(v_2)$ are contained in $N(v_1)$, v_1 is visited collaterally when v_2 is covered. Trivially, it is easy to see that if all targets have the same radius, then O2 is only verified if v_2 is exactly v_1 (the targets are perfectly coincident).

2.3.2 Chromosome encoding and Fitness function

The chromosome is a representation of a solution for CETSP. Each chromosome is an ordered sequence of genes, where each represents a *turning point*. Note that a turning point p_i of the target v_i is the point where the path crosses the neighborhood of target v_i. Hence, the coordinates (x_i, y_i) of a generic turning point p_i are unfixed and can vary between chromosomes in the population. Thus, each chromosome has its turning point p_i relative to the target v_i, which provides more heterogeneity among individuals in the population. Since the chromosome describes an acceptable solution to the problem, the size of a chromosome $|c|$ is equal to the number of targets plus the depot, i.e., $|c| = |N| + 1$. We show an example of a chromosome in fig. 2.2.

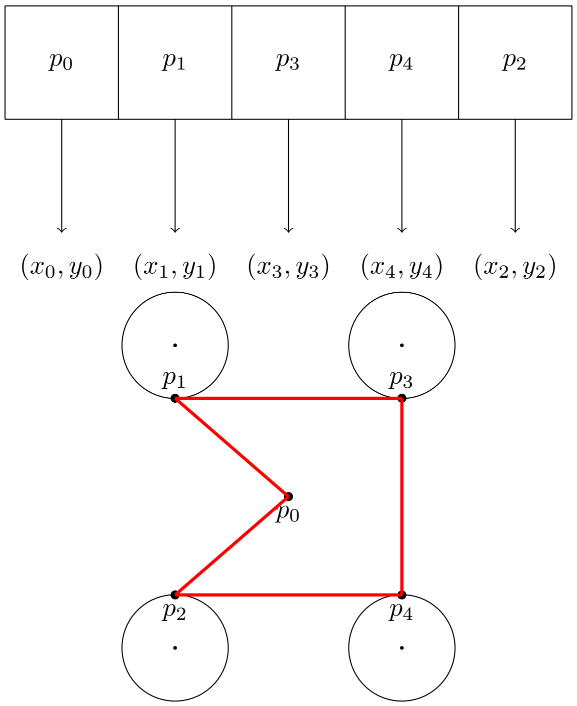


Figure 2.2: Example of chromosome with the corresponding turning points. [Di Placido et al., 2021]

A fitness function accurately evaluates each chromosome. Trivially, the fitness function is based on the total tour length $w(T)$ corresponding to the turning point sequence T. Better solutions are associated with lower fitness values, which means a tour of shorter length.

2.3.3 Initial Population

The population of each generation has a fixed number of chromosomes pop_{size}. We generate the initial population by defining pop_{size} chromosomes randomly. Specifically, for each target $v_i \in N$, we identify a turning point p_i. The location of this point is chosen randomly on the circumference C_{v_i}. Then, an eligible tour T is obtained by considering a random permutation of these turning points with the depot p_0 at the beginning and end of the tour. This process is repeated until the entire population is filled. Finally, the improvement procedures described in 2.3.7 are applied to all individuals.

2.3.4 Generation of new individuals

The population of each generation is obtained through the generation of two population halves in two different steps. Let $P = \{c_0, c_1, ..., c_k\}$ be the current population, where $c_i \in P$ is a generic chromosome. The elements of P are sorted in ascending order by fitness value. By doing so, the top ranges of the population contain the best individuals for that population. From P, we preserve the first $\frac{|P|}{2}$ chromosomes for the next generation P', where $|P|$ is assumed to be even. Then, to complete P', the second half of the population is obtained by applying the crossover operator described in sec. 2.3.5. This operator works by taking into account two chromosomes to define a third. The chromosomes on which the crossover is applied are selected: let $c_i \in P/P'$ be a generic chromosome from the remaining population. A chromosome $c_j \in P'$ is randomly chosen to pair with c_i, such that the crossover is applied on the pair (c_i, c_j). This process is repeated $\forall c_i \in P/P'$.

2.3.5 Crossover

The crossover operator is inspired by the two-point crossover proposed in [Booker, 1987]. Specifically, given two chromosomes c_1, c_2 we define a third c_{child} from the combination of the two. The composition of the new chromosome is done as follows: two random indices i and k are computed, where i is the starting index and k is the number of genes to be considered. Starting with i, we select k genes from c_1 and return them to c_{child}. Then, starting with $(i+k) mod \, |c_1|$, we select k genes from c_2 and return them to c_{child}. In this procedure, in case some genes inherited from c_2 were already present in the new chromosome, they are discarded. By already present genes we mean turning points associated with targets with which a turning point is already associated. Finally, if $|c_{child}| = |c_1| = |c_2|$, then c_{child} is an eligible tour, otherwise it is incomplete. The remaining turning points are included by determining the minimum cost inclusion as follows: $\forall p_j \in c_2 : p_j \notin c_{child}$, we determine the segment $\overline{p_i, p_{i+1}} : p_i, p_{i+1} \in c_{child}$ for which the distance $d(p_i, p_j) + d(p_j, p_{i+1})$ is minimal. Therefore p_j is included in the tour between p_i and p_{i+1}. An example of how the operator works is shown in Figure 2.3. The crossover terminates when $|c_{child}| = |c_1| = |c_2|$ and returns c_{child} as output. Alg. 1 presents the pseudocode of the operator.

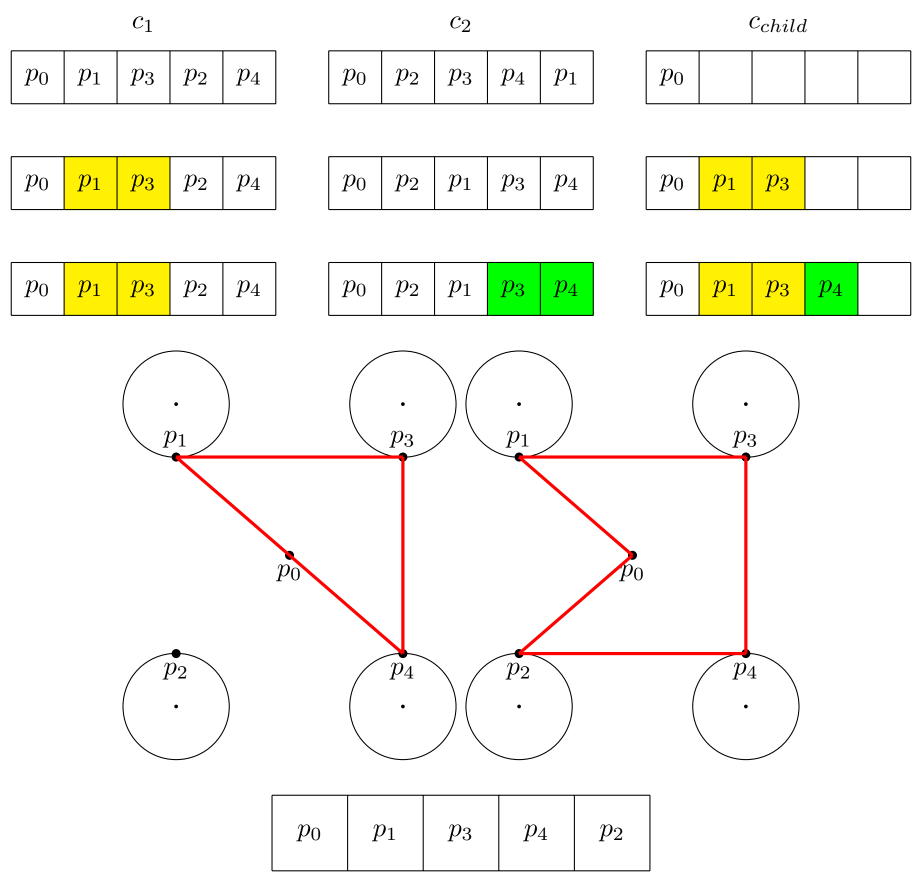

c_1
c_2
c_{child}
p_0 p_1 p_3 p_2 p_4
p_0 p_2 p_3 p_4 p_1
p_0
p_0 p_1 p_3 p_2 p_4
p_0 p_2 p_1 p_3 p_4
p_0 p_1 p_3
p_0 p_1 p_3 p_2 p_4
p_0 p_2 p_1 p_3 p_4
p_0 p_1 p_3 p_4
p_1 p_3 p_1 p_3
p_0 p_0
p_2 p_4 p_2 p_4
p_0 p_1 p_3 p_4 p_2

Figure 2.3: Illustration of crossover operator. In this example, the starting index $i = 1$ and the number of genes $k = 2$. From c_1 we take the genes from p_1 to p_3, while from c_2 we consider from p_3 to p_4. Since p_3 was already included from c_1, it is not repeated. Then, c_{child} is incomplete, since the point p_2 is not visited and it is necessary to add it into the sequence. The operator determines the nearest segment to the unvisited point, in this case $\overline{p_4 p_0}$, and adds the unvisited point between the vertices of the segment, as showed on the right picture. Finally, the completed c_{child} is $\{p_0, p_1, p_3, p_4, p_2\}$. [Di Placido et al., 2021]

2.3.6 Mutation

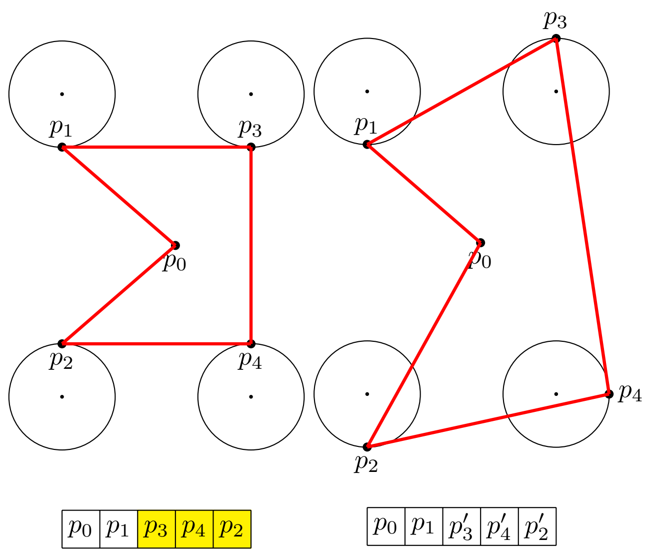
The mutation operator aims to diversify individuals and prevent premature convergence. In our implementation, it is applied with a certain probability *prob* on every chromosome c in the first half of the population. The value of *prob* corresponds to the number of generations without improvement divided by 100. Thus, the more the genetic algorithm iterates without improving the incumbent, the higher the probability of the mutation operator occurring on the

Algorithm 1 Crossover operator pseudocode

```
1: procedure CrossoverOperator(Chromosome $c_1$, Chromosome $c_2$)
2:     Let $size$ be the length of a chromosome
3:     Let $visited$ be an array of booleans that tells if $v \in N$ is covered
4:     $c_{child} \leftarrow \emptyset$
5:     $i = random(0, size - 1)$
6:     $k = random(1, size - 1)$
7:     for $j$ from 0 to $k$ do
8:         $c_{child} \leftarrow gene_{(i+i) mod\ size}$ from $c_1$
9:         $visited_j = true$
10:    end for
11:    $i = i + k$
12:    for $j$ from 0 to $k$ do
13:        $c_{child} \leftarrow gene_{(i+j) mod\ size}$ from $c_2$ s.t. $visited_j = false$
14:        $visited_j = true$
15:    end for
16:    if $|c_{child}| < size$ then
17:        for all $v_j$ unvisited do
18:            find $p_i, p_{i+1} \in c_{child} : d(p_i, p_j, p_{i+1})$ is minimum, with $p_j \in c_2$
19:            $c_{child} \leftarrow p_j$ between $p_i, p_{i+1}$
20:            $visited_j = true$
21:        end for
22:    end if
23:    return $c_{child}$
24: end procedure
```

chromosomes. The best individual is omitted from this process.

The operator chooses an arbitrary interval $[i_1, i_2]$ computed randomly. Then, every turning point $p_j \in c$ that belongs to the interval is replaced by a random point in the neighborhood $N(v_j)$. An example of the application of this operator is illustrated in fig. 2.4.



p_0	p_1	p_3	p_4	p_2

p_0	p_1	p_3'	p_4'	p_2'

Figure 2.4: Example of mutation. On the left we have a tour T. The random interval computed is $[2, 4]$, corresponding to the turning points p_3, p_4, p_2. Finally, the new points are generated randomly and included into the sequence. [Di Placido et al., 2021]

2.3.7 Improvement Procedures

Enhancement procedures are performed when a chromosome is generated or modified. We defined three different approaches to chromosome enhancement: a 2opt local search to improve the tour crossing sequence, a second-order cone programming algorithm, and a bisection algorithm, called 3Alg, used to adjust the position of turning points, given a fixed sequence. The 3Alg procedure was introduced to speed up the algorithm's execution, as we were initially using SOCP to improve the position of the points in a sequence. 2opt and 3Alg are applied sequentially, while SOCP is applied occasionally to a chromosome in the population. Using these procedures allows us to refine the individuals generated during execution and permits us to classify our algorithm as a *memetic algorithm* [Moscato et al., 1989].

2opt

The procedure takes the points in pairs, excluding the depot, and considers the swap. If by making the swap we have an improvement on the length of the tour, then it is made effective, and the procedure restart. 2opt ends when all possible pairs have been considered. In Alg. 2 we show the pseudocode. It is important to note that applying 2opt, we do not change the position of the turning points, but if the sequence is changed, the modification is recorded on the chromosome.

Algorithm 2 2opt pseudocode

```
 1: procedure 2OPT(T)
 2:     loop:
 3:     for $i$ from 1 to $|T| - 2$ do
 4:         for $j$ from $i + 1$ to $|T| - 1$ do
 5:             $T' \leftarrow \emptyset$
 6:             $T' \leftarrow T$, from 0 to $i - 1$
 7:             $T' \leftarrow T$, from $i$ to $k$ in reverse order
 8:             $T' \leftarrow T$, from $k + 1$ to $|T|$
 9:             if $w(T) < w(T')$ then
10:                 $T = T'$
11:                 goto loop
12:             end if
13:         end for
14:     end for
15:     return $T$
16: end procedure
```

Second-Order Cone Programming Algorithm

Given an eligible tour T, the second-order cone programming algorithm (SOCP) improves the length of the tour by changing the position of the visited turning points without impacting the sequence with which the targets are traversed. Specifically, [Mennell, 2009] showed how, when a tour sequence is fixed a priori, CETSP matches the touring Steiner zones problem. The above problem can be formulated as a second-order cone programming and solved in polynomial time. We report and describe the formulation proposed in [Mennell et al., 2011] and used in the GA below. The goal of SOCP is to determine the optimal position of the turning points p_i, $i = 0, ..., |N| + 1$, given a fixed sequence. p_0 and $p_{|N|+1}$ correspond to the depot. The location p_i is defined by the variables x_i, y_i. Let $\overline{x_i}, \overline{y_i}$ be the coordinates of the target covered by the turning point p_i. The SOCP is formulated as follows:

$$min \sum z_i \tag{2.1}$$

s.t.

$$w_i = x_i - x_{i+1} \qquad \forall i \in 0, ..., |N| \tag{2.2}$$
$$u_i = y_i - y_{i+1} \qquad \forall i \in 0, ..., |N| \tag{2.3}$$
$$s_i = \overline{x_i} - x_i \qquad \forall i \in 0, ..., |N| \tag{2.4}$$
$$t_i = \overline{y_i} - y_i \qquad \forall i \in 0, ..., |N| \tag{2.5}$$
$$z_i^2 \geq w_i^2 - u_i^2 \qquad \forall i \in 0, ..., |N| \tag{2.6}$$
$$s_i^2 + t_i^2 \leq r_i^2 \qquad \forall i \in 0, ..., |N| \tag{2.7}$$
$$z_i \geq 0 \qquad \forall i \in 0, ..., |N| \tag{2.8}$$
$$w_i, u_i, s_i, t_i, x_i, y_i, free \qquad \forall i \in 0, ..., |N| \tag{2.9}$$

We remind that the depot position is fixed, then $x_0 = \overline{x_0}$ and $y_0 = \overline{y_0}$. The variable z_i represents the Euclidean distance between the turning point p_i and its successor p_{i+1}. The objective function minimizes the sum of the distances between the turning points visited by the tour. The variables w_i and u_i are used to compute the Euclidean distance z_i, while the variables s_i and t_i are used in the constraints (2.4) and (2.5) to ensure that the location of the turning points p_i is within the neighborhood $N(v_i)$. Solving to the optimum, the SOCP provides the best possible location of the turning points. Nevertheless, solving the above formulation on all chromosomes can increase the computation time. Therefore, we only solve the SOCP for one random individual per generation. Thus because we have noticed empirically that the application on an individual can speed up the time of convergence of the algorithm without impacting the execution time. We use a greedy algorithm, 3Alg, described in the next section for the remaining ones.

3Alg

The 3Alg improvement procedure is a greedy heuristic that determines the location of turning points, given a fixed sequence of targets. The 3Alg is a variant of the Steiner point selection presented in [Wang et al., 2019]. The method considers a triple of turning points $\{p_i, p_j, p_k\}$ and the target v_j associated with the intermediate point p_j; the points p_i, p_k are stable, while p_j is the point to be repositioned. The goal is to reposition p_j such that the distance of the points is minimized and p_j belongs to the neighborhood of v_j. Formally, we want to find the coordinates of p_j that can minimize $d(p_i p_j) + d(p_j p_k)$ and such that $d(v_j p_j) \leq r_j$. From this, if we consider the segment $\overline{p_i, p_k}$, we have two different possible scenarios as shown in Fig.2.5: the segment either traverses $N(v_j)$ or is outside of it. In the case where the segment intersects the circle at two points A, B, we can choose any point on the segment $\overline{AB}$. In our implementation, the point p_j is chosen randomly between A and the midpoint of $\overline{AB}$. In the case where the segment does not intersect the circle, then p_j must be located on the circumference C_{v_j}. To identify the position of p_j, we used the bisection algorithm proposed by [Wang et al., 2019]. Unlike them, 3Alg is applied iteratively on all the turning points of the tour following the sequence of traversals. In

[Wang et al., 2019], the algorithm is applied by alternating even and odd points and vice versa. From our experiments, we noticed that the results obtained from 3Alg are comparable with those obtained with SOCP. Moreover, we noticed that there is no apparent difference with the version proposed by [Wang et al., 2019].

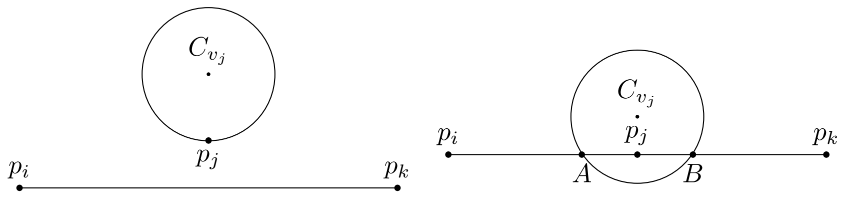


Figure 2.5: The two different scenarios for 3Alg. On the left $\overline{p_i p_k}$ does not traverse C_{v_j}, while on the right it does. [Di Placido et al., 2021]

2.3.8 Stopping Criteria

The genetic algorithm terminates when it reaches a number of iterations equal to it_{size} or when there is no improvement on the best fitness value after a specific number of generations it_{impr}. At the end of execution, the best chromosome in the current population is returned as output.

2.4 Metrics

In this section, we propose two new metrics and show their interaction. In addition, we revised the *overlap ratio* metric used in previous work on CETSP, for which we discovered an error in the calculation of its value. The purpose is to identify problem features related to the difficulty of solving the problem and the structure of the solution

2.4.1 Overlap Ratio

The *overlap ratio* (OR) is a metric introduced by [Mennell, 2009] that estimates the possible improvement of a solution over a TSP solution. Below, we report the definition presented in [Mennell, 2009]: "for a given CETSP instance, let $l_{contain}$ be the length of a side of the smallest square containing all n disks. The overlap ratio is the ratio of r to $l_{contain}$." Formally, The overlap ratio is the ratio between the average value of the radii of all targets and $l_{contain}$, i.e. $OR = \frac{\sum_{i=1}^{n} r_i}{l_{contain}}$. Figure 2.6 shows an example of the same instance with two different overlap ratios. We can see that large radii of the targets result in more significant improvements in the route length. While studying and calculating the OR on the benchmark instances of CETSPs, we noticed that our values differ with those proposed in [Mennell, 2009, Coutinho et al., 2016] on several instances. The reason is that, in these works, the OR is computed considering a rectangle that always has a vertex in (0,0). It is not always accurate since we can have points with negative coordinates or points with positive coordinates but different from (0,0). Therefore, we corrected the OR calculation by setting the smaller side of the rectangle containing all targets as $min(\Delta x, \Delta y)$, where Δx and Δy are the differences between the highest and smallest value of the coordinates. The correct values are shown in bold in the tables we present in the following sections.

Despite this, OR does not provide a good prediction of the difficulty of the instance. Therefore, we provide two complementary metrics that can measure the complexity of an instance through its morphology and the shape of its solution.

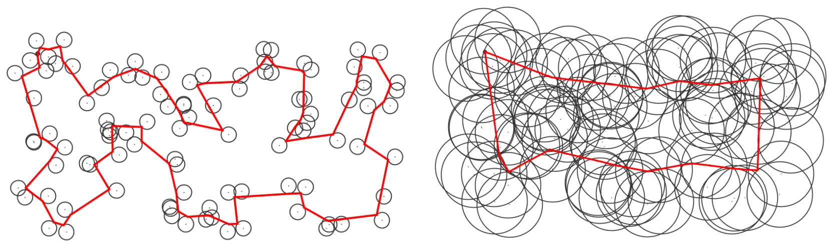


Figure 2.6: Instance *kroD100* with two different overlap ratio. On the left, OR is 2.00%, while on the right is 10.00% [Di Placido et al., 2021]

2.4.2 TSPDegree

The *TSPDegree* (TSPD) measurement is a metric based on the distances between targets and their radii. The goal is to understand what impact the radii have on the path based on the distance between targets. Let v_i, v_j be two targets, we calculate $m_{ij} = \frac{d(v_i,v_j)}{r_i+r_j}$. The value of the ratio m_{ij} provides a measure of the condition of the two targets, specifically whether they are overlapped or not:

- if $m_{ij} = 0$, the targets are concentric;
- if $0 < m_{ij} < 1$, the targets overlap and their intersection is nonempty,
- if $m_{ij} \geq 1$, the targets have no areas in common (empty intersection).

In the case where $m_{ij} > 1$, we set it equal to 1. From here, if we extend this concept to the k targets closest to a target v_i, it is possible to estimate the degree of overlap relative to v_i, which we will denote as M_i^k. The value of M_i^k is computed as $M_i^k = \frac{\sum_{j \in S_i^k} m_{ij}}{k}$, where S_i^k is the set of k targets closest to v_i. Thus, the value of $TSPD(k)$ corresponds to the average of M_i^k of all targets.

This measure allows us to estimate the composition of a CETSP instance at a morphological level: If the value of TSPD is low, it means that there is a high degree of overlap between the disks of the targets, so the main problem is to determine the positions of the turning points. On the contrary, if the value of TSPD is high, the instance is more sparse, i.e., the instance is traceable to a TSP. An explanatory example is illustrated in the figure 2.7

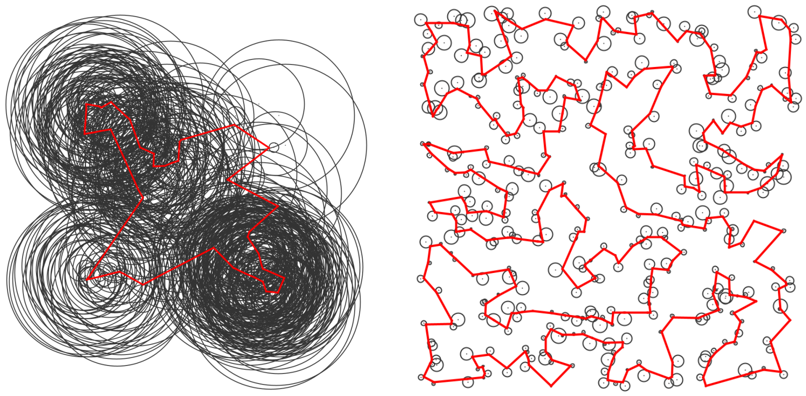


Figure 2.7: Example of two instances with different TSPDegree. On the left, the instance *team5_499rdmRad* (OR 2.00%) with TSPD equals to 8.49%, while on the right the instance *rd400rdmRad* (OR 0.99%) with TSPD equals to 96.91% [Di Placido et al., 2021]

2.4.3 OverlappingCenter

A second metric we propose is Overlapping Center (OC), a metric by which we estimate the difficulty of an instance based on the conformation of the solution. For example, if we consider an instance with a high number of targets, we might trivially assume that it is computationally difficult to solve. However, this is not always the case. Figure 2.8 shows the instance *team6_500* and an admissible solution of it in red. As can be seen, although the number of targets is 500, the solution consists of only six segments. The above instance is solved to the optimum by the exact approach proposed by [Coutinho et al., 2016] in 0.43 seconds.

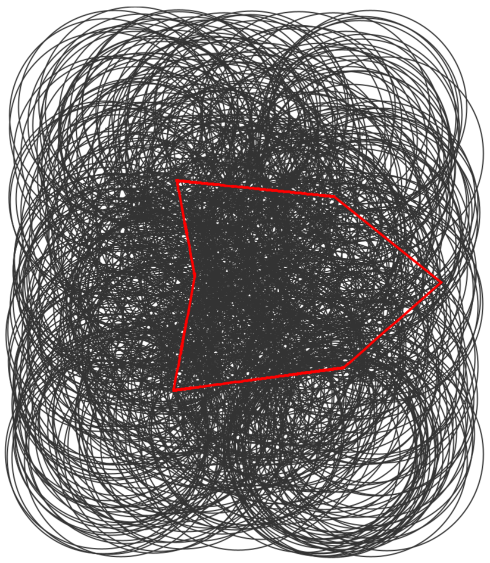


Figure 2.8: Instance *team6_ 500*, with OR 27.06%, TSPD 6.87% and OC 1.20%. [Di Placido et al., 2021]

The goal of the OC metric is to evaluate the overlap between target disks. The value of OC is computed by constructing a solution heuristically in which the smallest possible number of segments forms the tour T. The tour T is defined as follows. Starting from an empty T, for each target v_i, we consider the number of targets covered by its center and sort all targets in descending order by this value. Then, we insert in T the turning points corresponding to the centers of the targets following this order until all targets are covered. Note that, when the procedure stops, T can include a number of turning points less than $|N|$ since several targets can be covered by turning points associated with neighboring targets. The crossing sequence of T is identified by applying the 2opt procedure described in 2.3.7. Finally, excess turning points are removed as follows: starting with a triplet of turning points $\{p_i, p_j, p_k\}$ visited sequentially in T, and the target v_j associated with p_j, if the segment $\overline{p_i, p_k}$ crosses $N(v_j)$, then p_j is removed from T. This step is repeated for all consecutive triplets of T and continues to iterate until no further turning points are removed. At the end of this process, the number of segments is minimized, resulting in a tour eligible T s.t. $|T| \leq |N|$. The value of the OC measure is given by $OC = \frac{|T|}{|N|}$ and varies in the range $(0, 1]$. Trivially, OC can never be equal to 0 because at least one turning point must be visited. On the contrary, OC is equal to 1 when the tour has the same number of turning points as the number of targets.

2.5 Computational results

This section is devoted to computational experiments and results. It is divided into three macro categories. In the section 2.5.1 we evaluate the performance of our genetic algorithm by comparing it with state-of-the-art approaches. Then, in the 2.5.2 section, we focus on analyzing the metrics and their validity. Finally, in the 2.5.3 section, we present a real-world application related to solar panel diagnostics.

2.5.1 GA performance

We performed several tests to verify the effectiveness and efficiency of the GA. The algorithm was written in Java language (JDK 13.0), and the tests were performed on a Windows 10 Home machine equipped with an AMD Ryzen 7 3750H 2.30 GHz processor and 16 GB of RAM. The parameters described in 2.3 were set as follows: the number of individuals per population pop_{size} was set equal to 50; the number of generations it_{size} was set equal to 1000, and the number of generations without improvement to stop the GA it_{impr} was set equal to $\frac{it_{size}}{20}$. This configuration has been defined empirically and has been demonstrated to be that one with the just compromise between times of computation and the obtained solution. In addition, the results described below regarding the genetic algorithm are obtained by a single run of the approach.

The benchmark instances used for the tests are those proposed in [Mennell, 2009]. These are divided into three groups, named *Teams*, *Geometric*, and *TSPLIB*. The *Teams* instances are six instances whose names begin with *team*, along with the *bonus1000* instance. The number of targets is given in the name, excluding the depot. For example, the instance *team2_200* has 201 targets, including the depot. All instances are characterized by constant radius targets.

The *Geometric* instances are fixed-radius instances where the morphology of the instance refers to geometric shapes. This set includes the five *rotating-Diamonds*, five *concentricCircles*, and nine *bubbles*, for a total of 19 instances. The *TSPLIB* instances are created from the TSPLib library. The name infers the number of targets, including the depot. For these instances, we have three different fixed-radius scenarios: low overlap ratio (0.02), medium overlap ratio (0.1), and high overlap ratio (0.3).

For the instances *Teams* and *TSPLIB* there are variable radius counterparts, identified by the suffix *rdmRad* at the end of the instance name.

Comparison with optimal solutions

[Coutinho et al., 2016] identified several optimal solutions for the instances considered by our tests through the B&B algorithm. The approach was coded in the C++ language and tested on a Linux Mint 13 machine with an Intel Core i7 3.40 GHz processor and 16GB of RAM. The table 2.1 shows the GA solutions compared to the optimal ones. We compared only the solutions for which the B&B does not exceed the time limit of four hours (14400 seconds). The table is structured as follows: the first column shows the name of the instances; the second and third columns show the value of the solution produced by the GA and the computational time to find it (in seconds), while the fourth and fifth columns present the value of the exact solution and the computational time of the exact approach. Finally, the last column reports the percentage GAP, calculated using the formula: $\frac{GA-Opt}{Opt} \times 100$, where Opt is the value of the optimal solution and GA is the value of the solution found by the GA. Instances are grouped based on radii. Specifically, we identify three different groups: *Varied overlap ratios*, which includes instances with fixed radii for all targets but with different overlap ratios between instances; *Overlap Ratio*, which comprises instances where the radii are modified to obtain a specific OR value; and *Arbitrary Radius*, which contains instances where the targets have variable radii.

Instance	**GA**		**Opt**		**GAP**
	Value	*Time (s)*	*Value*	*Time (s)*	
		Varied overlap ratios			
bubbles1	**349.13**	2.21	**349.13**	0.10	0.00%
bubbles2	**428.28**	2.22	**428.28**	0.22	0.00%
bubbles3	**529.96**	7.29	**529.96**	193.12	0.00%
concentricCircles1	**53.16**	0.94	**53.16**	5.18	0.00%
rotatingDiamonds1	**32.29**	0.80	**32.29**	0.09	0.00%
rotatingDiamonds2	**140.48**	1.94	**140.48**	730.07	0.00%
team1_100	**307.34**	3.86	**307.34**	9.61	0.00%
team2_200	**246.68**	23.75	**246.68**	0.72	0.00%
team6_500	**225.22**	518.12	**225.22**	0.43	0.00%
		Overlap ratio 0.1			
d493	101.31	345.37	**100.72**	53.28	0.59%
kroD100	**89.67**	3.92	**89.67**	1.86	0.00%
lin318	1404.89	74.77	**1394.63**	8541.19	0.74%
rat195	68.14	21.31	**67.99**	17.32	0.22%
		Overlap ratio 0.3			
d493	**69.76**	233.54	**69.76**	0.32	0.00%
dsj1000	**199.95**	1181.44	**199.95**	0.75	0.00%
kroD100	**58.54**	2.20	**58.54**	0.07	0.00%
lin318	**765.96**	29.69	**765.96**	0.24	0.00%
pcb442	**83.54**	121.23	**83.54**	0.31	0.00%
rat195	**45.70**	6.49	**45.7**	0.13	0.00%
rd400	**224.84**	67.12	**224.84**	0.33	0.00%
		Arbitrary Radius			
rat195rdmRad	**68.22**	1.35	**68.22**	5.16	0.00%
team1_100rdmRad	**388.54**	1.84	**388.54**	269.31	0.00%
team3_300rdmRad	**378.09**	3.75	**378.09**	682.39	0.00%

Table 2.1: Comparison with optimal solutions.

GA is able to identify all optimal solutions across 23 instances, except for three instances where the overlap ratio is fixed at 0.1: *d493*, *line318*, and *rat195*. The solutions produced are still reasonably good for these instances, with a maximum gap of 0.74%.

Comparison with heuristic approaches

We compared the solutions of our GA with state-of-the-art approaches: the double-loop hybrid algorithm (HA) presented by [Yang et al., 2018]; the adaptive heuristic approach ((lb/ub)Alg) proposed by [Carrabs et al., 2020], and the variable neighborhood search heuristic (SZVNS) provided by [Wang et al., 2019]. Each algorithm was tested on its machine, which we report below:

- The HA experiments were conducted on a PC equipped with an Intel Core i5-4590 3.3 GHz CPU and 4GB of RAM;
- The (lb/ub)Alg trials were carried out on a PC equipped with an Intel i5 2.3 GHz as CPU and 8GB of RAM,
- SZVNS simulations were performed on a PC with a 2.6 GHz CPU and 4GB of RAM.

We report the results in the tables 2.2, 2.3, and 2.4, for which we apply the same instance categorization as applied for the table 2.1. The tables are organized as follows: the first column reports the name of the instances; in the second column, we record the best result found among the considered approaches; in the remaining four columns, we display the data for the solutions provided by GA, HA, (lb/ub)Alg and SZVNS respectively. In each of the columns for the approaches, we list the value of the solution, the computation time in seconds (*Time*), and the percentage gap from the best solution (GAP). The value of the percentage gap is calculated through the formula: $\frac{UB-Best}{Best} \times 100$, where UB is the value of the approach we are considering, and $Best$ is the value of the best solution found. Finally, in the last three rows of the tables, we report for each algorithm: the average of the values (Mean), the number of best solutions found (#Best), and the number of best solutions found only by the considered approach (#U.Best).

Instance	**Best**	**GA**			**HA**			**(lb/ub)Alg**			**SZVNS**		
		$w(T)$	*Time*	*GAP*	$w(T)$	*Time*	*GAP*	$w(T)$	*Time*	*GAP*	$w(T)$	*Time*	*GAP*
bonus1000	387.13	411.85	8786.91	6.39%	408.44	1508.10	5.50%	**387.13**	116.63	0.00%	403.06	1109.67	4.11%
bubbles1	349.13	**349.13**	2.21	0.00%	**349.14**	55.40	0.00%	**349.13**	11.42	0.00%	**349.14**	0.24	0.00%
bubbles2	428.28	**428.28**	2.22	0.00%	**428.28**	97.50	0.00%	**428.28**	22.68	0.00%	**428.28**	1.09	0.00%
bubbles3	529.96	**529.96**	7.29	0.00%	532.28	146.20	0.44%	**529.96**	130.17	0.00%	532.21	2.51	0.42%
bubbles4	805.46	**805.46**	27.84	0.00%	832.27	202.90	3.33%	805.56	130.11	0.01%	825.33	9.53	2.47%
bubbles5	1038.16	**1038.16**	45.07	0.00%	1067.96	234.90	2.87%	1061.64	116.63	2.26%	1073.43	37.37	3.40%
bubbles6	1229.66	**1229.66**	176.16	0.00%	1394.14	284.90	13.38%	1313.02	217.20	6.78%	1263.68	14.00	2.77%
bubbles7	1607.31	**1607.31**	809.15	0.00%	1735.38	521.00	7.97%	1650.04	304.46	2.66%	1639.33	50.29	1.99%
bubbles8	1946.72	**1946.72**	1230.27	0.00%	2120.98	412.50	8.95%	2021.27	416.30	3.83%	1972.99	110.30	1.35%
bubbles9	2259.22	**2259.22**	2655.50	0.00%	2456.27	415.50	8.72%	2413.31	296.14	6.82%	2330.31	168.18	3.15%
chaoSingleDep	1039.61	**1039.61**	10.69	0.00%	1042.82	201.40	0.31%	**1039.61**	89.69	0.00%	**1039.63**	12.67	0.00%
concentricCircles1	53.16	**53.16**	0.94	0.00%	**53.16**	18.50	0.00%	**53.16**	6.43	0.00%	**53.16**	0.10	0.00%
concentricCircles2	153.13	**153.13**	2.77	0.00%	**153.13**	40.10	0.00%	154.81	51.96	1.10%	154.88	0.27	1.14%
concentricCircles3	270.04	**270.04**	2.79	0.00%	271.08	7.60	0.39%	272.69	362.24	0.98%	272.49	1.28	0.91%
concentricCircles4	452.64	**452.64**	12.56	0.00%	455.62	105.70	0.66%	466.52	98.86	3.07%	461.36	5.82	1.93%
concentricCircles5	632.99	**632.99**	26.04	0.00%	647.64	141.20	2.31%	659.36	351.21	4.17%	647.84	10.06	2.35%
rotatingDiamonds1	32.29	**32.29**	0.80	0.00%	**32.29**	16.60	0.00%	**32.29**	7.06	0.00%	32.39	0.12	0.31%
rotatingDiamonds2	140.48	**140.48**	1.94	0.00%	**140.48**	58.90	0.00%	**140.48**	176.13	0.00%	**140.48**	0.54	0.00%
rotatingDiamonds3	380.88	**380.88**	10.60	0.00%	382.50	192.10	0.43%	**380.89**	615.47	0.00%	380.89	8.81	0.00%
rotatingDiamonds4	770.66	**770.66**	41.85	0.00%	777.05	257.10	0.83%	772.00	271.20	0.17%	**770.68**	18.49	0.00%
rotatingDiamonds5	1510.75	**1510.75**	376.41	0.00%	1530.31	765.40	1.29%	1531.74	210.99	1.39%	1510.88	53.62	0.01%
team1_100	307.34	**307.34**	3.86	0.00%	**307.34**	122.60	0.00%	**307.34**	78.86	0.00%	**307.34**	3.20	0.00%
team2_200	246.68	**246.68**	23.75	0.00%	247.48	336.20	0.32%	**246.68**	36.06	0.00%	**246.69**	16.21	0.00%
team3_300	464.20	**464.20**	99.30	0.00%	466.12	366.20	0.41%	476.43	56.88	2.63%	465.80	31.48	0.34%
team4_400	685.52	**685.52**	428.43	0.00%	686.76	474.80	0.18%	702.69	120.25	2.50%	698.05	43.47	1.83%
team5_499	700.50	**700.50**	995.23	0.00%	711.14	447.20	1.52%	708.45	828.33	1.13%	703.38	88.17	0.41%
team6_500	225.22	**225.22**	518.12	0.00%	227.50	819.90	1.01%	**225.22**	8.15	0.00%	226.18	719.32	0.43%
Mean	690.63	691.55	603.66	0.24%	720.65	305.57	2.25%	708.51	190.06	1.46%	701.11	93.22	1.09%
#Best			26			7			12			8	
#U.Best			13			0			1			0	

Table 2.2: Computational results for the varied overlap ratios instances.

Instance	Best	GA			HA			(lb/ub)Alg			SZVNS		
		w(T)	*Time*	*GAP*	*w(T)*	*Time*	*GAP*	*w(T)*	*Time*	*GAP*	*w(T)*	*Time*	*GAP*
Overlap ratio 0.02													
d493	202.23	**202.23**	864.47	0.00%	203.84	435.90	0.80%	205.39	247.04	1.56%	205.74	69.72	1.74%
dsj1000	938.71	**938.71**	20325.36	0.00%	949.37	841.50	1.14%	955.57	998.99	1.80%	943.83	151.77	0.55%
kroD100	159.04	**159.04**	9.05	0.00%	159.05	85.10	0.01%	160.09	167.14	0.66%	**159.04**	9.34	0.00%
lin318	2838.54	**2838.54**	174.28	0.00%	2883.17	245.00	1.57%	2902.53	292.90	2.25%	2842.32	31.17	0.13%
pcb442	322.54	**322.54**	826.50	0.00%	325.05	381.40	0.78%	377.51	805.44	17.04%	325.02	52.88	0.77%
rat195	158.32	**158.32**	55.23	0.00%	158.79	162.30	0.30%	166.51	569.55	5.17%	160.06	11.16	1.10%
rd400	1032.04	**1032.04**	686.27	0.00%	1041.77	308.00	0.94%	1085.75	662.60	5.20%	1039.77	27.70	0.75%
Overlap ratio 0.1													
d493	100.72	101.31	345.37	0.59%	101.75	689.00	1.02%	**100.72**	33.87	0.00%	102.92	142.94	2.18%
dsj1000	374.06	376.87	3951.05	0.75%	380.59	1372.90	1.75%	**374.06**	83.00	0.00%	393.06	356.79	5.08%
kroD100	89.67	**89.67**	3.92	0.00%	**89.67**	125.30	0.00%	**89.67**	193.44	0.00%	89.92	1.99	0.28%
lin318	1404.89	**1404.89**	74.77	0.00%	1410.25	444.50	0.38%	1405.07	26.50	0.01%	1414.66	65.98	0.70%
pcb442	145.82	**145.82**	466.16	0.00%	148.74	653.00	2.00%	146.03	224.95	0.14%	152.73	54.99	4.74%
rat195	68.14	**68.14**	21.31	0.00%	68.24	269.30	0.15%	68.14	95.19	0.00%	68.32	9.47	0.26%
rd400	458.41	**458.41**	612.93	0.00%	469.19	560.80	2.35%	460.21	101.27	0.39%	474.78	96.98	3.57%
Overlap ratio 0.3													
d493	69.76	**69.76**	233.54	0.00%	70.40	983.40	0.92%	**69.79**	2.72	0.04%	69.90	58.07	0.20%
dsj1000	199.95	**199.95**	1181.44	0.00%	202.94	1766.60	1.50%	**199.95**	6.89	0.00%	203.07	316.41	1.56%
kroD100	58.54	**58.54**	2.20	0.00%	59.03	189.80	0.84%	**58.54**	0.37	0.00%	**58.54**	4.58	0.00%
lin318	765.96	**765.96**	29.69	0.00%	770.08	589.80	0.54%	**765.96**	1.21	0.00%	766.16	49.01	0.03%
pcb442	83.54	**83.54**	121.23	0.00%	84.02	846.70	0.57%	**83.54**	0.58	0.00%	83.80	196.54	0.31%
rat195	45.70	**45.70**	6.49	0.00%	48.50	371.20	6.13%	**45.70**	0.33	0.00%	**45.70**	19.01	0.00%
rd400	224.84	**224.84**	67.12	0.00%	226.09	743.90	0.56%	**224.84**	4.19	0.00%	224.98	74.86	0.06%
Mean	463.88	**464.04**	1431.35	0.06%	469.07	574.54	1.15%	473.60	215.15	1.63%	467.82	85.78	1.14%
#Best			19			2			10			3	
#U.Best			9			0			2			0	

Table 2.3: Computational results for instances with different overlap ratio.

Instance	**Best**	**GA**			**HA**			**(lb/ub)Alg**			**SZVNS**		
		w(T)	*Time*	*GAP*	*w(T)*	*Time*	*GAP*	*w(T)*	*Time*	*GAP*	*w(T)*	*Time*	*GAP*
bonus1000rdmRad	932.07	**932.07**	480.23	0.00%	1001.10	1043.70	7.41%	955.41	728.55	2.50%	938.27	3.93	0.67%
d493rdmRad	134.28	**134.28**	5.58	0.00%	141.34	770.70	5.26%	134.74	93.34	0.34%	135.02	0.26	0.55%
dsj1000rdmRad	624.75	**624.75**	25.86	0.00%	659.64	1594.60	5.58%	625.92	286.23	0.19%	625.25	2.73	0.08%
kroD100rdmRad	141.83	**141.83**	6.97	0.00%	**141.84**	110.70	0.01%	142.39	46.25	0.39%	**141.83**	87.16	0.00%
lin318rdmRad	2047.42	**2047.42**	8.71	0.00%	2079.49	483.00	1.57%	2055.77	59.52	0.41%	2082.25	3.18	1.70%
pcb442rdmRad	220.00	**220.00**	16.16	0.00%	234.15	620.80	6.43%	220.44	153.38	0.20%	221.16	1.36	0.53%
rat195rdmRad	68.22	**68.22**	1.35	0.00%	**68.22**	427.10	0.00%	**68.22**	9.97	0.00%	**68.22**	10.57	0.00%
rd400rdmRad	1246.69	**1246.69**	701.06	0.00%	1252.22	277.30	0.44%	1305.46	1110.35	4.71%	1257.73	0.67	0.89%
team1_100rdmRad	388.54	**388.54**	1.84	0.00%	390.23	129.70	0.43%	**388.54**	20.80	0.00%	**388.54**	3.89	0.00%
team2_200rdmRad	613.66	**613.66**	16.04	0.00%	624.79	211.80	1.81%	616.82	209.44	0.51%	626.90	0.93	2.16%
team3_300rdmRad	378.09	**378.09**	3.75	0.00%	382.16	488.00	1.08%	378.51	16.65	0.11%	379.84	34.51	0.46%
team4_400rdmRad	1000.31	**1000.31**	317.04	0.00%	1020.16	348.50	1.98%	1025.76	658.03	2.54%	1006.71	2.62	0.64%
team5_499rdmRad	446.19	**446.19**	2.52	0.00%	458.35	874.90	2.73%	446.51	51.43	0.07%	**446.19**	5.05	0.00%
team6_500rdmRad	620.98	**620.98**	26.15	0.00%	672.36	701.20	8.27%	626.18	347.08	0.84%	621.99	19.77	0.16%
Mean	633.07	633.07	115.23	0.00%	651.86	577.29	3.07%	642.19	270.79	0.92%	638.56	12.62	0.56%
#Best			14			2			2			4	
#U.Best			10			0			0			0	

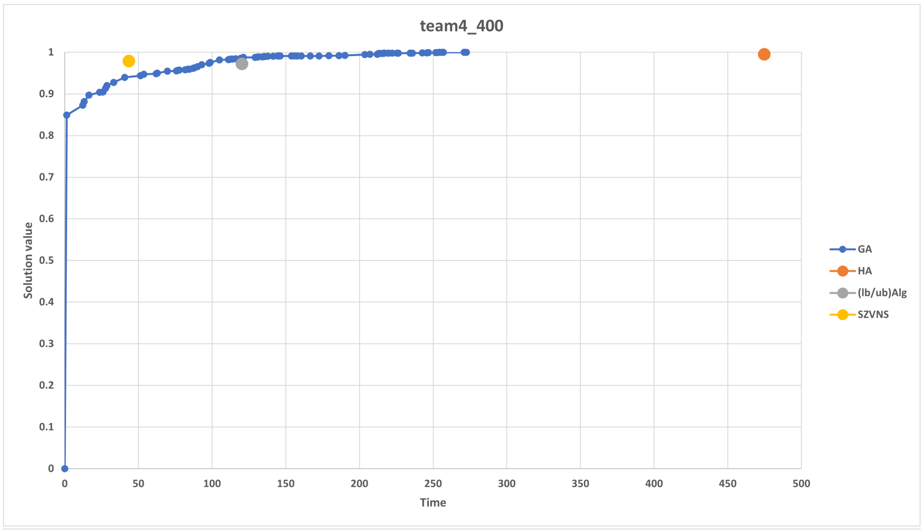
Table 2.4: Computational results for the arbitrary radii instances.

For the first group of instances, GA can identify 13 new best solutions out of 27 instances, while in the other 13, it still finds the best-known solution. Only in one case, GA performs worse than its counterparts. For the second group, GA finds 9 out of 21 new unique best solutions, while in 10 other instances, it identifies the best-known solution. For the last group of instances, GA can find 10 new unique best solutions out of 14 instances, while for the remaining 4, it identifies the best-known solution. The results clearly show that GA performs significantly better than existing approaches on all categories of instances in terms of solution quality, identifying the most significant number of unique best solutions and generating solutions with the smallest gap compared to the best-known solutions.

Overall, GA identified 32 new best solutions and 27 best-known solutions out of a total of 62 instances. GA produces an improvement in all categories of instances. In the *variable overlap ratio* instances, we obtain the most considerable improvement, obtaining solutions ranging from a minimum percentage gap of 0.18% to a maximum of 3.15% from the current best solution; on the *overlap ratios* we obtain an improvement ranging from a minimum of 0.04% to a maximum of 0.80% GAP. Finally, on the *arbitrary radius* instances, we span from a minimum of 0.11% to a maximum of 0.66% GAP.

As for computational times, overall, they are reasonable. The GA is much slower in the varied overlap ratio instances than the other approaches. On the other hand, GA is the second-fastest approach in arbitrary radii instances after SZVNS.

To further evaluate GA performance, we performed additional comparisons with other approaches. Specifically, we examined the speed of convergence of our approach by comparing it to the performance of others, noting how quickly GA achieves counterpart results. We considered a small set of instances for which GA takes more than 100 seconds to produce a solution, namely *team4_400*, *team5_499*, *bubbles7*, and *bubbles8*. We only show results for this small set w.l.o.g. since the behavior is the same in the other instances. The results are shown in the graphs in Figures 2.9 and 2.10, in which the computational time is shown on the abscissa and the solution value on the ordinate. As we can see, GA obtains the solutions of the other approaches in lesser or comparable times, except for SZVNS.



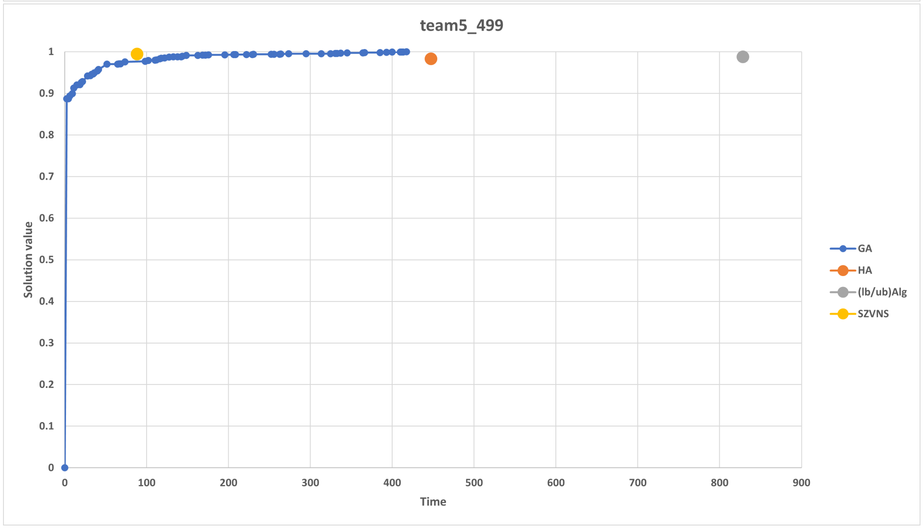


Figure 2.9: Time to target comparison of GA with respect to the other approaches for the instances team4_400 and team5_499. [Di Placido et al., 2021]

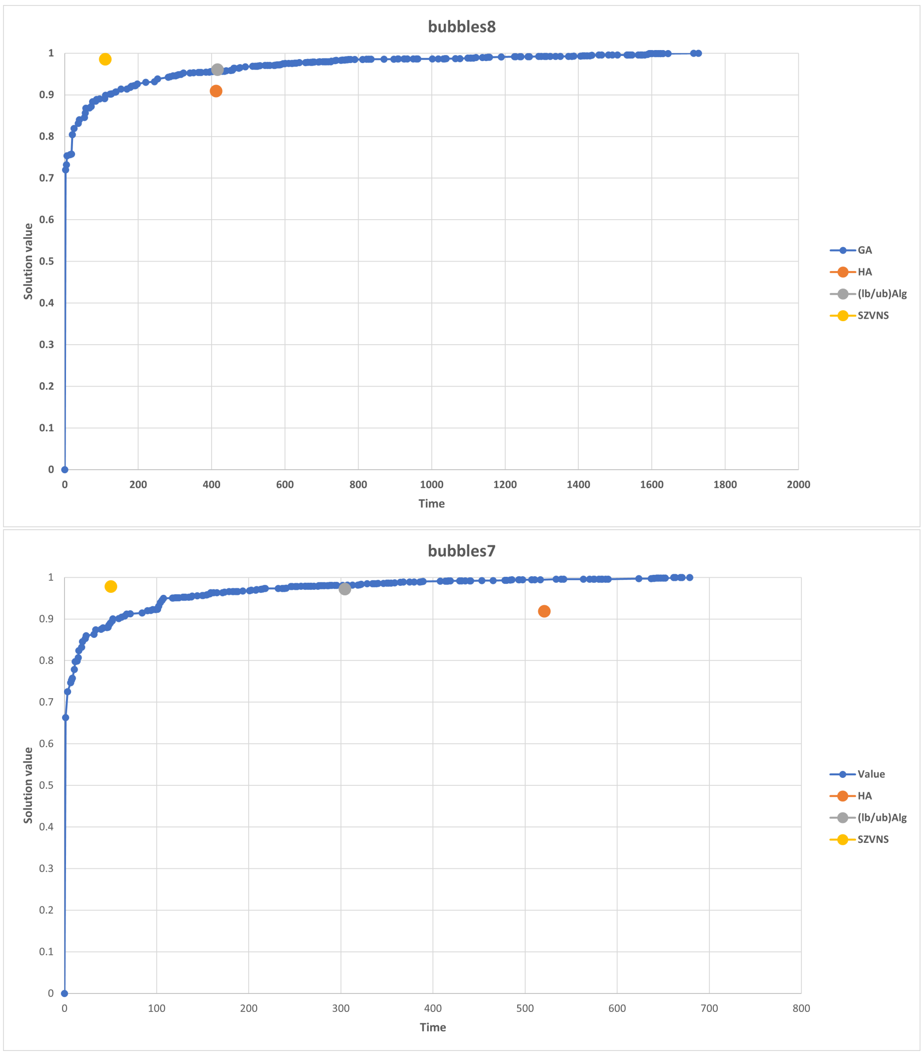


Figure 2.10: Time to target comparison of GA with respect to the other approaches for the instances bubbles7 and bubbles8. [Di Placido et al., 2021]

2.5.2 Metrics analysis

We conducted several analyses to verify the validity of the proposed metrics. We used the instances proposed by [Mennell, 2009] for these tests. The value of the metrics for each instance is reported in the tables 2.5, 2.6, and 2.7. For the calculation of the TSPD measure, we used $k = 5$.

Instance	$\|N\|$	**OR**	**TSPD**	**OC**
Varied overlap ratios (Teams and Geometrics)				
bonus1000	1000	12,26%	10,88%	1,70%
bubbles1	36	**11,11%**	78,70%	19,44%
bubbles2	76	**9,09%**	58,90%	15,79%
bubbles3	126	**7,69%**	57,28%	21,43%
bubbles4	184	**6,67%**	56,47%	15,22%
bubbles5	250	**5,88%**	55,99%	18,40%
bubbles6	324	5,26%	55,67%	15,43%
bubbles7	406	4,76%	55,44%	12,32%
bubbles8	496	**4,35%**	55,27%	13,31%
bubbles9	594	4,00%	55,14%	14,48%
chaoSingleDep	200	2,84%	52,78%	17,50%
concentricCircles1	16	**15,00%**	86,01%	37,50%
concentricCircles2	36	**7,50%**	89,99%	50,00%
concentricCircles3	60	**5,00%**	90,76%	45,00%
concentricCircles4	104	**3,75%**	90,16%	40,38%
concentricCircles5	148	**3,00%**	90,13%	36,49%
rotatingDiamonds1	20	**20,00%**	77,66%	30,00%
rotatingDiamonds2	60	**5,00%**	84,74%	18,33%
rotatingDiamonds3	180	**3,33%**	81,32%	10,56%
rotatingDiamonds4	320	1,43%	84,37%	8,13%
rotatingDiamonds5	680	1,11%	81,99%	5,00%
team1_100	100	9,33%	36,68%	15,00%
team2_200	200	20,06%	16,61%	4,50%
team3_300	300	7,02%	28,60%	11,00%
team4_400	400	5,01%	43,45%	17,00%
team5_499	499	2,00%	53,78%	23,05%
team6_500	500	27,06%	6,87%	1,20%

Table 2.5: Metrics values for the varied overlap ratios instances. The values in bold are the OR values that do not correspond to the ones proposed in the literature.

Instance	$\lvert N\rvert$	**OR**	**TSPD**	**OC**
Overlap ratio 0.02				
d493	492	2,00%	46,68%	19,51%
dsj1000	999	2,00%	43,67%	16,92%
kroD100	99	2,00%	93,08%	54,55%
lin318	317	2,00%	76,22%	32,18%
pcb442	441	2,00%	82,41%	39,91%
rat195	194	2,00%	96,40%	47,94%
rd400	399	2,00%	88,31%	46,37%
Overlap ratio 0.1				
d493	492	10,00%	10,08%	2,03%
dsj1000	999	10,00%	8,98%	1,40%
kroD100	99	10,00%	34,41%	14,14%
lin318	317	10,00%	19,94%	5,68%
pcb442	441	10,00%	19,09%	3,40%
rat195	194	10,00%	27,09%	8,25%
rd400	399	10,00%	23,94%	6,02%
Overlap ratio 0.3				
d493	492	30,00%	3,36%	0,61%
dsj1000	999	30,00%	2,99%	0,40%
kroD100	99	30,00%	11,47%	2,02%
lin318	317	30,00%	6,65%	0,95%
pcb442	441	30,00%	6,36%	0,68%
rat195	194	30,00%	9,03%	1,03%
rd400	399	30,00%	7,98%	1,75%

Table 2.6: Metrics values for instances with different overlap ratio

Instance	$\lvert N \rvert$	**OR**	**TSPD**	**OC**
		Arbitrary Radius		
bonus1000rdmRad	1000	6,13%	27,12%	11,20%
d493rdmRad	492	13,44%	9,89%	5,28%
dsj1000rdmRad	999	**12,47%**	9,66%	4,30%
kroD100rdmRad	99	4,03%	74,97%	31,31%
lin318rdmRad	317	10,05%	24,65%	12,62%
pcb442rdmRad	441	9,39%	26,14%	14,74%
rat195rdmRad	194	42,60%	8,65%	6,19%
rd400rdmRad	399	0,99%	96,91%	67,67%
team1_100rdmRad	100	7,69%	48,80%	26,00%
team2_200rdmRad	200	5,19%	64,44%	32,50%
team3_300rdmRad	300	23,70%	11,60%	5,33%
team4_400rdmRad	400	2,05%	79,72%	40,00%
team5_499rdmRad	499	20,14%	8,49%	4,01%
team6_500rdmRad	500	10,02%	22,99%	10,60%

Table 2.7: Metrics values for the arbitrary radii instances. The values in bold are the OR values that do not correspond to the ones proposed in the literature.

Note that several OR values differ from those presented in [Mennell, 2009, Coutinho et al., 2016], as we corrected the computational error. Values that differ are highlighted in bold.

The first analysis conducted aims to test whether there is a correlation between OR and the new TSPD and OC metrics. The figure 2.11 depicts the values of the metrics with respect to OR for each instance.

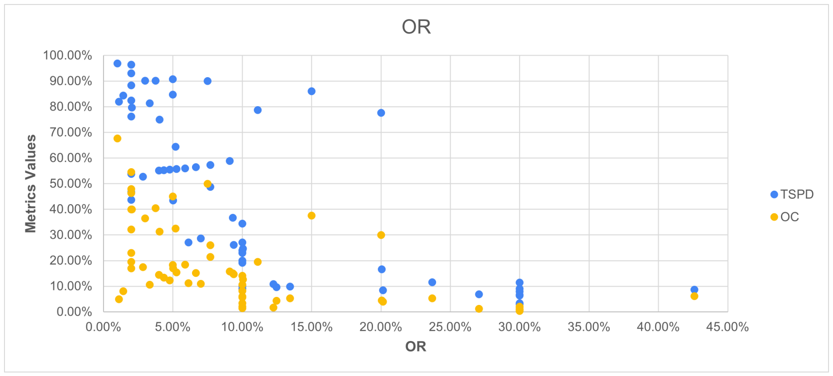


Figure 2.11: Values of the metrics TSPD and OC with respect to OR for each instance. On the x-axis, we have OR values, while on the y-axis, we have the values of TSPD and OC. [Di Placido et al., 2021]

TSPD and OC appear to have similar trends in most cases. In fact, in general, high values of TSPD seems to be correlated with high values of OC and low values of OR. It is reasonable since if the targets span the entire plane with little overlap (high value of TSPD), more segments are needed to reach them. An illustrative example of this case is shown in Fig. 2.12.

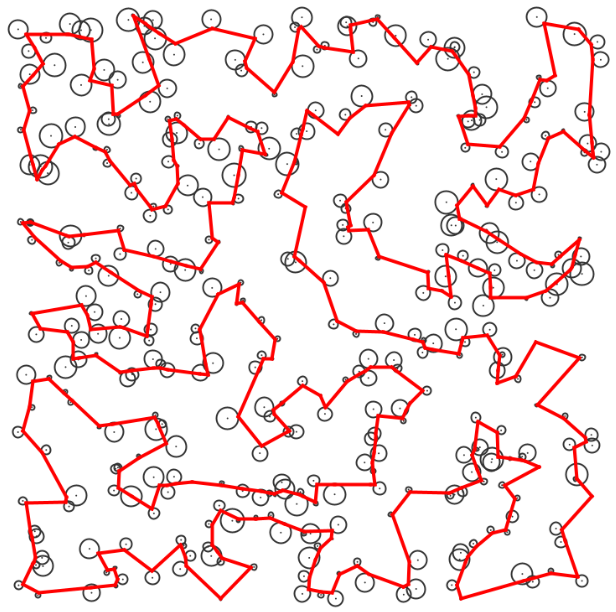


Figure 2.12: Instance *rd400rdmRad*: OR 0.99%, TSPD 96.91%, OC 67.67%. [Di Placido et al., 2021]

A notable exception occurs on some instances with low OR values, specifically from 0% to 5%. These instances relate low to medium OC values and high TSPD values. Based on this, the instances present targets far apart, and their solution is composed of a small number of segments. These instances are specific to instances that have several dense groups of targets that are far apart and that follow a geometric shape in their placement, i.e., *Geometrics* instances. Two examples are shown in Fig. 2.13. Hence, comparing the two instances in the figure shows that both have a low OR value, 2.05%, and 5%, respectively. Despite this, we can graphically see that the solution of *rotatingDiamonds2* is much easier to find than that of *team4_400rdmRad*. It is related to the route's greater regularity (and the position of the targets), a characteristic apprehended by the low OC value but not by the OR value.

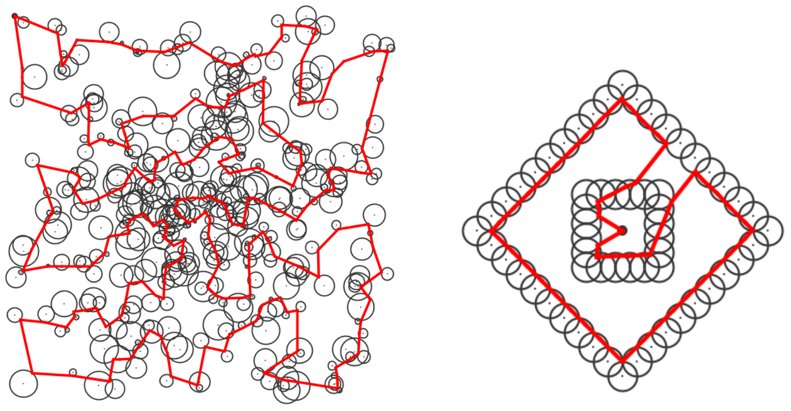


Figure 2.13: Example of two instances with high TSPD value and mid-low OC value: *team4_400rdmRad* (OR 2.05%, TSPD 79.72%, OC 40.00%) on the left and *rotatingDiamonds2* (OR 5.00%, TSPD 84.74%, OC 18.33%) on the right. [Di Placido et al., 2021]

A second conducted analysis aims to verify a correlation between the metrics and the computation times employed for the different algorithms. As shown above, for several instances, the optimal solution is known [Coutinho et al., 2016]. We examined the latter to see a common pattern in the metrics. The values of the metrics of the instances solved to the optimum are reported in the table 2.8, organized as follows: the first column reports the name of the instance, while the following two columns record the computational times and the number of targets of the instance. Finally, the last three columns list the values of the metrics.

Instance	**Time**	$\|N\|$	**OR**	**TSPD**	**OC**
concentricCircles1	5.18	16	15.00%	86.01%	37.50%
rotatingDiamonds1	0.09	20	20.00%	77.66%	30.00%
bubbles1	0.10	36	11.11%	78.70%	19.44%
rotatingDiamonds2	730.07	60	5.00%	84.74%	18.33%
bubbles2	0.22	76	9.09%	58.90%	15.79%
kroD100	0.07	99	30.00%	11.47%	2.02%
kroD100	1.86	99	10.00%	34.41%	14.14%
team1_100	9.61	100	9.33%	36.68%	15.00%
team1_100rdmRad	269.31	100	7.69%	48.80%	26.00%
bubbles3	193.12	126	7.69%	57.28%	21.43%
rat195	0.13	194	30.00%	9.03%	1.03%
rat195	17.32	194	10.00%	27.09%	8.25%
rat195rdmRad	5.16	194	42.60%	8.65%	6.19%
team2_200	0.72	200	20.06%	16.61%	4.50%
team3_300rdmRad	682.39	300	23.70%	11.60%	5.33%
lin318	0.24	317	30.00%	6.65%	0.95%
lin318	8541.19	317	10.00%	19.94%	5.68%
rd400	0.33	399	30.00%	7.98%	1.75%
pcb442	0.31	441	30.00%	6.36%	0.68%
d493	0.32	492	30.00%	3.36%	0.61%
d493	53.28	492	10.00%	10.08%	2.03%
team6_500	0.43	500	27.06%	6.87%	1.20%
dsj1000	0.75	999	30.00%	2.99%	0.40%

Table 2.8: Metric values for the instances solved to optimality

We can observe that, in general, high OR values correspond to greater ease of resolution. Regarding the TSPD metric, we can see that on large instances, the value of TSPD (and also OC) is low, as expected. However, we notice how they take relatively long to resolve if we focus on the smaller instances. For example, referring to the instances *team1_ 100rdmRad* and *bubbles3*, for both the TSPD value is around 50%. It means that the solution of these instances is very far from being a TSP on targets, a condition that occurs when the value of TSPD is high. Hence, although the TSP and CETSP are both NP-Hard, the CETSP is more complex since it is the combination of defining the route and choosing the position of the turning points. On the other hand, the value of TSPD is not low enough to get an "easy" instance to solve, as in the case of instances with a low OC value. Another ambiguous case is *rotatingDiAmonds2* for which we did not find an explanation for the relatively high computational time.

Finally, OC values follow an increasing trend, i.e., instances in which the tour consists of a few segments are resolved in less time. Furthermore, OC provides the most meaningful information regarding the computational complexity of the instance. For example, if we examine instances *dsj1000* with OC value 0.40%, and *concentricCircles1* with OC value 37.50%, these are solved in 0.75 and 5.18 seconds, respectively. Hence, although *dsj1000* has 999 targets, this one is solved in less time than *concentricCircles1*, which has only 16 targets. The two instances are shown graphically in Fig. 2.14. We can conclude that the number of segments composing a solution discriminates on the complexity of an instance of CETSP than the size of the instance itself.

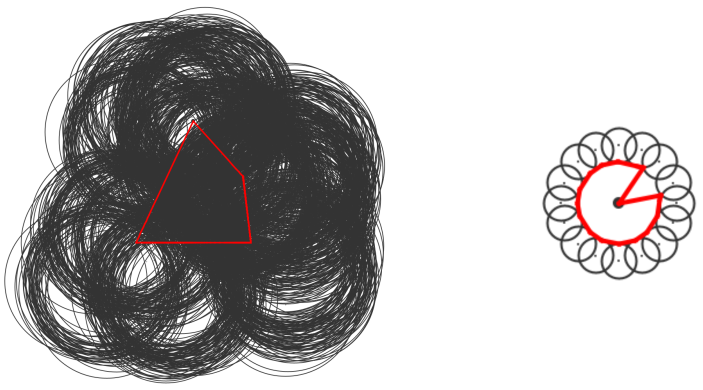


Figure 2.14: Instance *dsj1000* (OR 30%, TSPD 2.99%, OC 0.40%) on the right and instance *concentricCircles1* (OR 15%, TSPD 86.01%, OC 37.50%). [Di Placido et al., 2021]

Related to this issue, we performed further analysis by considering the approaches presented in 2.5.1. The goal is to determine whether a specific metric is related to the computational difficulty of the instance. To determine this, we defined five different classes for each metric: low, low-mid, mid, mid-high, and high. The table 2.9 shows the ranges of each class for each metric.

Metrics	Classes	Range
TSPD	1 - Low	0%-20%
	2 - Low-Mid	20%-40%
	3 - Mid	40%-60%
	4 - Mid-High	60%-80%
	5 - High	80%-100%
OC	1 - Low	0%-5%
	2 - Low-Mid	5%-15%
	3 - Mid	15%-20%
	4 - Mid-High	20%-40%
	5 - High	40%-100%
OR	1 - Low	0%-5%
	2 - Low-Mid	5%-10%
	3 - Mid	10%-15%
	4 - Mid-High	15%-30%
	5 - High	30%-100%

Table 2.9: Metrics classes based on range

The measures span different ranges. for OR, we considered the estimate proposed by [Mennell, 2009, Coutinho et al., 2016], in which $OR = 0.02$ is defined as *low*, $OR = 0.1$ as *mid*, and $OR = 0.3$ as *high*. On the other hand, for TSPD the values increase homogeneously in the interval [0,100], so it is reasonable to consider an equal division of classes. Finally, the OC values are concentrated under the threshold of 30%. Therefore, the *high* class ranges above 30%, while the rest are generated based on the density of instances in each class. Next, we categorized the instances by class for each metric and calculated the average gap from the best-known solution and the average computational time of the approaches relative to the class of the metric.

The categorization is shown in the table 2.10. The table is defined on three macro rows, one for each metric. Each row is organized as follows: the first column reports the classes of the metrics, while the columns for the approaches provide the average times and gaps. The computational times have been normalized to the number of targets of each instance to have a more homogeneous estimation. Finally, the columns *Mean* exhibit the total average of the approaches relative to the class.

TSP Degree										
	Times					Gaps				
Classes	GA	HA	(lb/ub)Alg	SZVNS	Mean	GA	HA	(lb/ub)Alg	SZVNS	Mean
$TSPD_1$	0,96	1,71	0,09	0,27	0,76	0,41%	1,98%	0,05%	1,08%	0,88%
$TSPD_2$	0,33	1,33	0,61	0,06	0,58	0,00%	3,32%	0,87%	0,94%	1,28%
$TSPD_3$	2,35	1,03	0,68	0,11	1,04	0,00%	3,34%	1,96%	1,34%	1,66%
$TSPD_4$	0,27	1,03	0,79	0,17	0,57	0,00%	0,90%	0,95%	0,54%	0,60%
$TSPD_5$	0,50	0,88	2,12	0,05	0,89	0,00%	0,60%	3,12%	0,70%	1,11%
Overlapping Center										
	Times					Gaps				
Classes	GA	HA	(lb/ub)Alg	SZVNS	Mean	GA	HA	(lb/ub)Alg	SZVNS	Mean
OC_1	1,16	1,68	0,11	0,30	0,81	0,48%	2,02%	0,11%	1,17%	0,95%
OC_2	0,67	1,30	0,69	0,09	0,69	0,00%	3,34%	1,16%	0,87%	1,34%
OC_3	2,42	1,06	0,76	0,07	1,08	0,00%	2,20%	1,49%	1,27%	1,24%
OC_4	0,52	1,00	1,09	0,13	0,69	0,00%	0,99%	2,55%	0,65%	1,05%
OC_5	0,59	0,77	2,50	0,04	0,98	0,00%	0,39%	2,98%	0,96%	1,08%
Overlap Ratio										
	Times					Gaps				
Classes	GA	HA	(lb/ub)Alg	SZVNS	Mean	GA	HA	(lb/ub)Alg	SZVNS	Mean
OR_1	2,06	0,91	1,38	0,14	1,12	0,00%	2,05%	3,23%	0,89%	1,54%
OR_2	0,21	1,07	1,15	0,04	0,62	0,00%	2,47%	1,30%	1,11%	1,22%
OR_3	1,19	1,43	0,40	0,18	0,80	0,55%	2,42%	0,17%	1,67%	1,20%
OR_4	0,24	1,51	0,14	0,33	0,56	0,00%	1,03%	0,04%	0,24%	0,33%
OR_5	0,28	1,93	0,01	0,18	0,60	0,00%	1,38%	0,01%	0,27%	0,42%

Table 2.10: Mean computing times and gaps of the heuristic approaches for each class of the metrics

In general, the values of the gaps for the OR metric follow a decreasing trend, supporting the hypothesis that instances with high OR values are easier to solve. Nevertheless, if we observe the computational times, the trend is inconstant. On the contrary, TSPD and OC have more homogeneous trends. For both metrics, the most difficult class is the third one, i.e., the *mid* class for both metrics. Thus, the approaches tend to have more difficulty when defining the tour and determining the location of the turning points have the same impact on the solution implementation, i.e., when the targets are arranged heterogeneously on the plane, and it is necessary to identify a tour with an intermediate number of segments to traverse them all. This classification includes all those instances that are neither too simple at the tour level (i.e., a fairly reasonable number of segments are required), nor too similar to a TSP. An example is illustrated in figures 2.15.

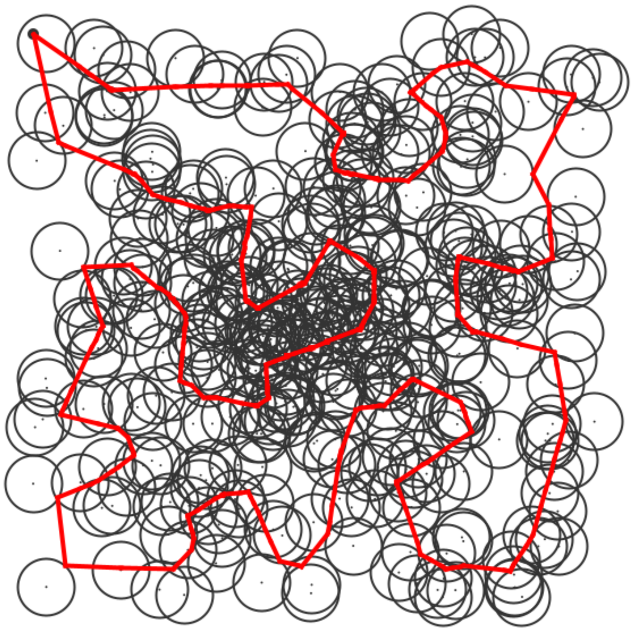


Figure 2.15: Instance *team4_400* with TSPD value equal to 43.45% (3rd class) and OC value equals to 17.00% (3rd class). [Di Placido et al., 2021]

2.5.3 Solar Panels Diagnostic

In this section, we present a real case study related to the diagnostic reconnaissance of solar panels. Specifically, in this case, we plan to use a drone equipped with a thermal camera to perform diagnostic reconnaissance tours to determine the operational status of photovoltaic fields. Therefore, it is necessary to determine the tours of the drone.

To simplify the structure of a photovoltaic field, we can say that the solar panels are connected in series through a structure called *String*. A set of Strings is connected in parallel to an inverter. The purpose of the inverter is to transform the direct current coming from the panels into alternating current. The maintenance of photovoltaic systems to ensure their proper functioning can be carried out through drones capable of taking thermal images. The thermography of a panel provides information about the operating temperature of the entire panel and the cells that compose it. Many conditions must be met in order to use a thermal photo for diagnosis:

- The photo must perfectly frame the panel, as shown in the figure 2.16, i.e., the position where to take the photo with respect to the inclination and the altitude must be suitable for the correct identification.
  - This is necessary to guarantee the image processing algorithm a correct recognition and to allow it to analyze it automatically.
- Photos should be captured as quickly as possible to ensure the validity of the diagnosis.
  - To verify the correct operation of the individual panels, it is necessary to compare the thermographs of the panels belonging to the same inverter. Through this comparison, we can understand if there are overloads on some panels of the same String or compare different strings with the same degree of energy absorption. The change of atmospheric agents or temperature changes affects the thermal images. So, if environmental conditions changes occur, and the drone goes too long in the tour, the comparison would be compromised.
- The available tour identification time is limited.
  - Thermal images should be taken in stable weather conditions. The presence of atmospheric agents such as wind or clouds impacts temperatures, and electricity production, resulting in inaccurate analysis. Keeping in mind that the calculation of the target area associated with each panel depends on the position of the sun, which constantly changes over time, it is necessary to have an algorithm fast enough to define the tour, to ensure the launch of the drone as soon as the weather conditions are stable.

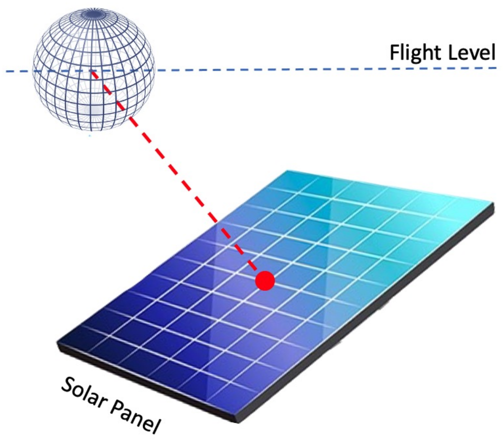


Figure 2.16: This image shows how to identify the area in which to take a thermal photo of the solar panel. From the center of the panel, trace the normal line to the panel up to the pre-established flight level. A sphere with a radius depending on the type of camera used is determined at the point of intersection between the normal line and the plane of flight. The intersection between the sphere and the plane of flight gives the circle to take a thermal image. It is essential to point out that the sphere's position and its radius can vary based on the position of the sun and the panel inclination. [Di Placido et al., 2021]

We modeled this scenario with a CETSP, whose resolution returns the drone's path to take thermal images. The implementation details and how we generated the instance are outside the scope of this dissertation. They cannot be described in detail as they were developed for a private partner company and are subject to disclosure agreements. In compensation, we report in figures 2.17 an example of an instance obtained from 48 solar panels. The instance files contain the positions of the targets with their rays (red circles). Note that the position is based on the position of the sun and the inclination of the panel. From this, the circles can translate with respect to the position of the panels. The position of the photovoltaic panels (blue squares) has been provided for graphical reasons.

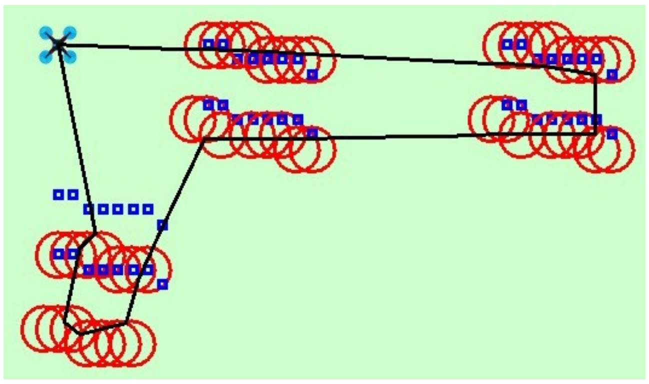


Figure 2.17: Example instance consisting of 48 solar panels connected to the same inverter and divided into 6 Strings. The circles in red represent the target areas associated with the photovoltaic panels represented by blue squares. Note that the circles' position can be shifted to the panels' position because of the panels' inclination. The icon representing a drone shows the starting point of the drone's route. In black, the path associated with the solution obtained with the algorithm proposed in this work. [Di Placido et al., 2021]

From the application scenario just described, it was necessary to define an algorithm capable of solving the CETSP effectively and efficiently. The goal is to identify the shortest route to produce high-quality photos for all targets. Hence, in terms of time and space, the route length is critical, both for critical drone-related and weather issues. Having a short route allows us to reduce the drone battery's problems and reduce the possible error due to weather factors. Another possible solution to reduce the travel time can be to increase the drone's speed, yet this could negatively impact the quality of the images due to the lack of stability.

A third party provided the radii of the targets based on the flight height and the lens mounted on the drone. Before the GA development, the route was determined by solving a TSP on the centers of the circles. The savings achieved in route length with the GA application amounts to 15%. Also, the number of photos produced compared to the TSP route is the same. Regarding the quality of the photos, they turn out to be qualitatively comparable or even better than those produced with the previous approach. For disclosure reasons, we cannot provide graphical examples. As part of this project, it was necessary to develop an approach capable of producing convolution filters to identify artifacts or patterns present in an image. It is crucial to compare the thermographs of the panels connected to the same inverter to verify the correct functioning of the individual panels. The comparison is necessary to understand if there is an overuse of some panels connected to the same String or some Strings compared to other Strings with the same energy absorption by the inverter. In addition, the image analysis in question could be disrupted by anything from scratch or erosion of the panel itself. One of the basic techniques used for image processing is convolution and its inverse, deconvolution [Nussbaumer, 2012]. Convolution is a mathematical operation that, given two functions f and g as input, produces a third that expresses how the first one has been modified by the other. It has several applications such as image quality improvement, e.g., denoising, feature extraction problems, object detection and recognition, motion tracking, and many other problems related to automatic photo and video processing [Chen and Fomel, 2015, Bovik, 2010, Alpaydin, 2009]. In contrast, deconvolution is an algorithmic process used to reverse convolution effects on data. Specifically, it aims to identify the function h such that $f \times g = h$. One possible approach to solving this equation is through the convolution theorem.

Theorem 1 *The Fourier transform of a convolution of two functions is equal to the product of the Fourier transform of the two functions.*

Nevertheless, this strategy is not applicable since the function g may not be unique, the function h may contain zeros, and the data may be affected by noise. Therefore, while convolution is always possible, deconvolution is not always feasible. Several methods have been developed in order to calculate the best possible inverse convolution, such as: Van-Cittert-Deconvolution [Xu et al., 1994]; Wiener-Deconvolution [Dhawan et al., 1985]; Blind deconvolution [Michailovich and Tannenbaum, 2007]; Gaussian elimination [Zhao and Desilva, 1998]; Singular value decomposition (SVD) [Sadek, 2012]; Truncated singular value decomposition (TSVD) [Wu et al., 2017]. The above problem is modeled as a filter retrieval problem (FRP), in which the main goal is to identify the filter f' such that, starting from the original image I and the filtered image $f(I)$, $f'(I) = f(I)$. Most existing techniques in the literature can retrieve the filter through a dataset of possible filters. Others use linear programming or integer linear programming, neural networks, or genetic algorithms to generate filters that reproduce the considered transformation.

We formulated a method capable of recovering the filters applied during the convolution step from the original image and the filtered one. To do so, we propose a mathematical formulation capable of generating filters that minimize the Mean Absolute Error (MAE). We define the average of the differences between the output obtained and the one we are looking for as MAE. Relative to images, this corresponds to the average pixel-by-pixel differences in absolute value between the two images.

2.5.4 Problem definition and notations

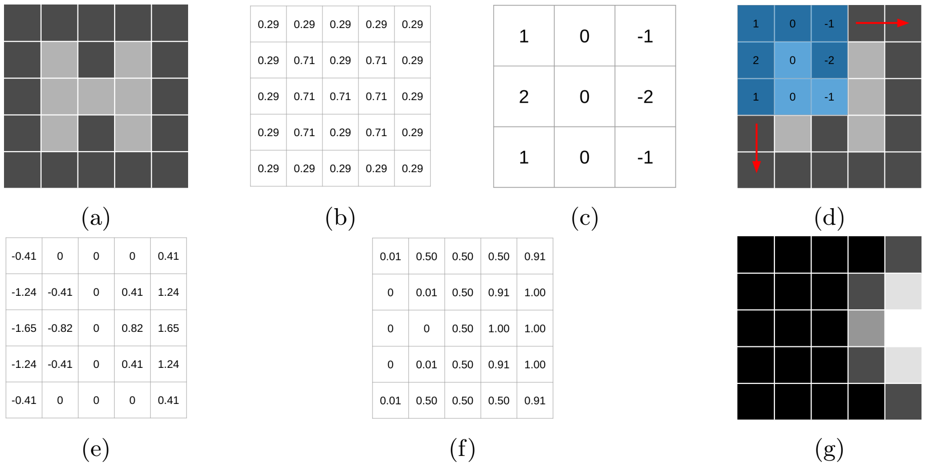
This section presents the notation used and the definition of the problem considered. We also provide an example describing the application of a filter to a specific image and MAE computation.

During our tests, w.l.o.g., we used grayscale images. In addition, we considered pixel values in a continuous range $[0, 1]$, although these are generally represented in the discrete range $[0, 255]$.

Given an image I and a filter $f = k \times h$, we denote by $I' = f(I)$ the image resulting from applying the filter f on the image I. The filter matrix is applied pixel by pixel to obtain the output image I'; specifically, the value of a generic pixel $I^{'}_{i,j}$ is calculated through the formula:

$$f(I_{i,j}) = I^{'}_{i,j} = \sum_{a=1}^{k}\sum_{b=1}^{h} I_{(i-\lfloor\frac{k}{2}\rfloor+a-1)(j-\lfloor\frac{h}{2}\rfloor+b-1)} \times f_{ab} \tag{2.10}$$

Fig. 2.18 shows an example of applying a filter f to an image I, with subsequent output I'. If h and k are even, the indices must be changed accordingly. In the continuation, we assume that these are odd.



(b)

0.29	0.29	0.29	0.29	0.29
0.29	0.71	0.29	0.71	0.29
0.29	0.71	0.71	0.71	0.29
0.29	0.71	0.29	0.71	0.29
0.29	0.29	0.29	0.29	0.29

(c)

1	0	-1
2	0	-2
1	0	-1

(e)

-0.41	0	0	0	0.41
-1.24	-0.41	0	0.41	1.24
-1.65	-0.82	0	0.82	1.65
-1.24	-0.41	0	0.41	1.24
-0.41	0	0	0	0.41

(f)

0.01	0.50	0.50	0.50	0.91
0	0.01	0.50	0.91	1.00
0	0	0.50	1.00	1.00
0	0.01	0.50	0.91	1.00
0.01	0.50	0.50	0.50	0.91

Figure 2.18: Example of filter applied to an image I. Figure 2.18a and 2.18b show the original 5×5 images and the matrix of pixels values, respectively. Figure 2.18c shows the filter matrix applied on the input image in Figure 2.18d. Figure 2.18g shows the resulting filtered images and 2.18e shows the resulting matrix of pixels values after the convolution. Figure 2.18f highlights the resulting matrix after the threshold cut application. [Capobianco et al., 2021]

Considering the example shown in Fig. 2.18, equation 2.10 can be exploded as follows:

$$\begin{aligned} I'_{i,j} = I_{i-1,j-1} \times f_{1,1} + I_{i-1,j} \times f_{1,2} + I_{i-1,j+1} \times f_{1,3} + \\ I_{i,j-1} \times f_{2,1} + I_{i,j} \times f_{2,2} + I_{i,j+1} \times f_{2,3} + \\ I_{i+1,j-1} \times f_{3,1} + I_{i+1,j} \times f_{3,2} + I_{i+1,j+1} \times f_{3,3} \end{aligned} \tag{2.11}$$

We use MAE in such a way that we compute the difference between two images. We calculate MAE as the average of the pixel-by-pixel differences in absolute values of the images. Formally, let I' and I'' be two images, the MAE between these images is calculated according to the formula:

$$MAE(I', I'') = \frac{\sum_{i=1}^{w(I')} \sum_{j=1}^{h(I')} |I^{'}_{i,j} - I^{''}_{i,j}|}{w(I') \times h(I')} \tag{2.12}$$

We propose a mathematical formulation capable of generating filters that minimize the Mean Absolute Error. Formally, given an image I and a filtered image $I' = f(I)$, we define a mathematical formulation that, from the two images, can define a filter f' that minimizes the MAE between I' and $I'' = f'(I)$. In real application cases, I' is not exactly equal to $f(I)$ but differs by a certain ϵ due to errors on the output image (e.g., noise, threshold). Our formulation is able to produce a filter f' that can be equivalent to f $(\epsilon = 0)$ or similar to less than a constant $(\epsilon > 0)$. We propose a linear programming formulation (LP) that does not consider any cutoffs or thresholds on the pixel values. Then, we propose an integer linear programming formulation (ILP) where the pixel values are in the interval [0,1]. Finally, we propose a third integer linear programming formulation that uses activation variables for image thresholding. Mathematically formulating the problem allows us to model the FRP to the optimum and certify the goodness of the obtained filter. In particular, it is possible to ascertain whether, given a transformation, it can be obtained using a f filter. The results obtained show that our approach is competitive with the state of the art for noise-free images. This work has been published in the journal Soft Computing [Capobianco et al., 2021]. In the following sections, we will briefly describe only the first two models, as they are not properly the scope of the proposed work. For more details, refer to [Capobianco et al., 2021].

2.5.5 Mathematical Formulation

LP-Model: Given an input image I, we want to identify

$$f' = \begin{bmatrix} x_{11} & \dots & x_{h1} \\ \vdots & \ddots & \vdots \\ x_{1k} & \dots & x_{hk} \end{bmatrix}$$

such that $MAE(I', f'(I))$ is minimal. For any $x_{ij} \in f'$, $x_{ij} \in [-\delta, \delta]$, with $\delta \in \Re$. Let $c_{i,j}$ be an auxiliary variable used to represent the value of pixel $i, j \in I$ after applying the filter f' on it. Let $e_{i,j}$ be a variable representing the error committed between the original filtered image and the one produced by f', i.e., the difference in absolute value of $c_{i,j}$ and $I^{'}_{i,j}$. The objective is:

$$Minimize \sum_{i,j} e_{i,j} \qquad \forall i, j \in I \tag{2.13}$$

s.t.

$$c_{i,j} = f'(I_{i,j}) \qquad \forall i,j \in I \qquad (2.14)$$
$$c_{i,j} - I^{'}_{i,j} \leq e_{i,j} \qquad \forall i,j \in I \qquad (2.15)$$
$$I^{'}_{i,j} - c_{i,j} \leq e_{i,j} \qquad \forall i,j \in I \qquad (2.16)$$
$$x_{i,j} \in [-\delta, \delta] \qquad \forall i,j \in I \qquad (2.17)$$
$$e_{i,j}, c_{i,j} \in \Re \qquad \forall i,j \in I \qquad (2.18)$$

The constraint 2.14 ties the pixel value $i,j \in I$ to be equal to $f'(I_{i,j})$, while the constraints 2.15 and 2.16 force $e_{i,j}$ to be larger than or equal to the error between $I^{'}_{i,j}$ and $c_{i,j}$ in absolute value.

This formulation provides the solution in polynomial time since the model is composed of linear equations with no integer variables. Nevertheless, applying a filter to an image may cause some pixel values to exceed the limits of the interval [0,1], as illustrated in Fig. 2.18e. Hence, the LP model generates an implicit error during filter identification due to the truncation of pixel values reflected in the MAE calculation, where we compare the already thresholded images and not the obtained values. We defined a second mathematical formulation of integer linear programming that considers this limitation.

ILP-Model: In this model, we introduced three new auxiliary variables, $c^{'}_{i,j}$, $c^{0}_{i,j}$, $c^{1}_{i,j}$, defined as follows:

$$c^{'}_{i,j} = \begin{cases} 0 & \text{if } c_{i,j} \leq 0 \\ c_{i,j} & \text{if } c_{i,j} \in (0,1) \\ 1 & \text{if } c_{i,j} \geq 1 \end{cases}$$

$$c_{i,j} \geq 1 \implies c^{1}_{i,j} = 1$$
$$c_{i,j} \leq 0 \implies c^{0}_{i,j} = 1$$

$c^{'}_{i,j}$ is a variable representing the value of the pixel $i,j \in I$ after applying the filter f' on it. $c^{0}_{i,j}$ and $c^{1}_{i,j}$ are two variables used to normalize the value of $c^{'}_{i,j}$. Let M be a sufficiently large number[1], we want to minimize the objective function (2.13) such that the constraint (2.14) and all subsequent constraints are satisfied:

[1]The value of M is computed by considering the maximum value that can be taken by a pixel after applying the f' filter on it. Based on the equation (2.10), to obtain the maximum value we will have that $\sum_{a=1}^{k}\sum_{b=1}^{h} I_{(i-\lfloor\frac{k}{2}\rfloor+a-1)(j-\lfloor\frac{h}{2}\rfloor+b-1)} = k \times h$ and $f_{ab} = \delta,$ for $a = 1,...,k$ and $b = 1,...,h$. Thus, $M = k \times h \times \delta$

$$c^0_{i,j} + c^1_{i,j} \leq 1 \qquad \forall i,j \in I \tag{2.19}$$
$$c_{i,j} + Mc^0_{i,j} \geq 0 \qquad \forall i,j \in I \tag{2.20}$$
$$c_{i,j} - Mc^1_{i,j} \leq 1 \qquad \forall i,j \in I \tag{2.21}$$
$$c^{'}_{i,j} \leq c_{i,j} + Mc^0_{i,j} \qquad \forall i,j \in I \tag{2.22}$$
$$c^{'}_{i,j} \geq c_{i,j} - Mc^1_{i,j} \qquad \forall i,j \in I \tag{2.23}$$
$$c^{'}_{i,j} \geq c^1_{i,j} \qquad \forall i,j \in I \tag{2.24}$$
$$c^{'}_{i,j} \leq 1 - c^0_{i,j} \qquad \forall i,j \in I \tag{2.25}$$
$$c^{'}_{i,j} - I^{'}_{i,j} \leq e_{i,j} \qquad \forall i,j \in I \tag{2.26}$$
$$I^{'}_{i,j} - c^{'}_{i,j} \leq e_{i,j} \qquad \forall i,j \in I \tag{2.27}$$
$$x_{i,j} \in [-\delta, \delta] \qquad \forall i,j \in I \tag{2.28}$$
$$e_{i,j}, c_{i,j}, c^{'}_{i,j} \in [0,1] \qquad \forall i,j \in I \tag{2.29}$$
$$c^0_{i,j}, c^1_{i,j} \in \{0,1\} \qquad \forall i,j \in I \tag{2.30}$$

The constraint 2.19 assures us that at most one variable between $c^0_{i,j}$ and $c^1_{i,j}$ is active, i.e., equal to 1. The constraints 2.20 and 2.21 force the variables $c^0_{i,j}$ and $c^1_{i,j}$ to be equal to 1 when $c_{i,j} \leq 0$ and $c_{i,j} \geq 1$ respectively. The constraint 2.22 guarantees that the value of $c^{'}_{i,j}$ will be less than or equal to the value of $c^1_{i,j}$ in the case where $c^0_{i,j}$ is inactive, i.e., equal to 0. The counterpart is the constraint 2.23, which ensures that the value of $c^{'}_{i,j}$ will be greater than or equal to the value of $c_{i,j}$ if $c^1_{i,j}$ is inactive. Finally, the constraints 2.24 and 2.25 respectively require $c^{'}_{i,j}$ to be equal to 1 if $c^1_{i,j} = 1$, and equal to 0 if $c^0_{i,j} = 1$.

The combination of the constraints ensures us that $c^{'}_{i,j}, c^1_{i,j}$ and $c^0_{i,j}$ are set correctly: the constraints 2.20 and 2.21 do not force the variables $c^1_{i,j}$ or $c^0_{i,j}$ to be inactive in the case where $c_{i,j} \in (0,1)$. In fact, this condition is satisfied by the constraints 2.22 and 2.23. Moreover, 2.24 and 2.25 force $c^{'}_{i,j}$ to be equal to 1 or 0, respectively, which may not happen by considering only 2.22 and 2.23.

Finally, the constraints 2.26 and 2.27 are used to greater or at most equal the error between $I^{'}_{i,j}$ and $c^{'}_{i,j}$ in absolute value.

2.5.6 Computational results

We ran several tests to verify the effectiveness of our methods. These were run on a machine running Windows 10 OS, with AMD Ryzen 7 3750 Processor 2.3 GHz and 16GB of RAM. The formulations were implemented in Java language (JDK 14.0) and were solved using CPLEX ver. 12.9. In our experiments, if not explicitly described, we considered a 3x3 matrix and a range $[-10, 10]$ as the filter size, $\delta = 10$. We restricted the filter size since most common kernels are of this size [Simonyan and Zisserman, 2014]. In addition, another crucial parameter is the range of values of the filter elements. As mentioned, we set the range to $[-10, 10]$, even though it is unlimited; yet, after some testing, we can say that the performance of our framework does not deteriorate by setting the range to $[-10, 10]$.

Several experiments have been conducted to study the behavior of the implemented models. In test A we analyzed the ability of the models to identify filters applied on the reference images. Test B focuses on the robustness of the proposed approaches, specifically their tolerance in images affected by noise. In Test C, we examine the ability of our formulations to define filters capable of emphasizing specific characteristics of an image. Test D aims to verify the impact of an image sample's choice on the solution produced. Finally, in test E, we check how the model behaves when the filter size and the range of values are varied.

In the following, we will briefly describe the results obtained from the tests without going into too much detail. For more information on the tests just mentioned, refer to the results section of [Capobianco et al., 2021].

Test A: We performed our experiments using benchmark images with geometric figures and images in this specific test. On these, filters known from image processing, such as Sobel or blur, and filters defined by us were applied (fig. 2.19). The results show that the ILP model identifies all applied filters to the optimum in 0.30 seconds, except for one case where the image produced by the obtained filter has an MAE of 0.2% compared to the original. At the same time, the LP model produces filters capable of producing filtered images with relatively low MAE.

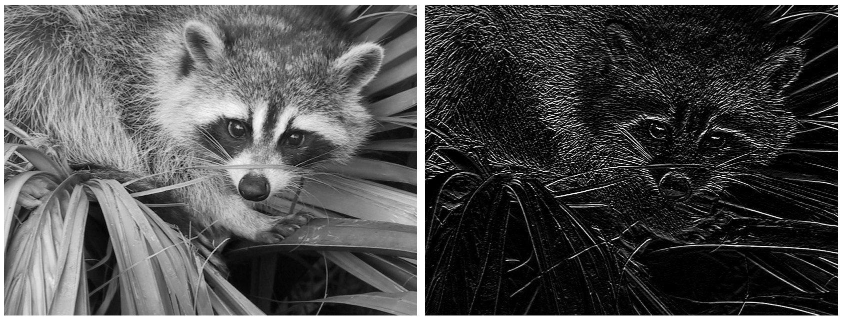

(a) Original (b) Filtered

Figure 2.19: Racoon Test images. 2.19a is the input image and 2.19b is the output images obtained applying a filter on it. [Capobianco et al., 2021]

Test B: For this test, we perturbed the filtered images with noise at different percentages, i.e., 1%, 3%, 5%, and 10%, to see how this impacts the solution produced by our models. The results show that the applied perturbations do not overly impact the results of the ILP model, which provides filters with

$MAE \leq 4\%$.

Test C: We examined the ability of our approaches to store filters capable of accentuating image features that were manually marked. To do this, we used samples of different shapes, such as hexagon, flower, rectangle, and triangle. For each of these, we manually emphasized features, such as edges or vertices. We divided the test into two steps: First, we produced the filter considering the images as a whole. In the second, we use only the feature sample we want to highlight to identify the filter (feature extraction). The results indicate that our approaches can reasonably identify all those filters that emphasize continuous areas of the image (e.g., edges). If we want to emphasize discontinuous areas, such as vertices, our approaches approximate them to continuous. Finally, LP and ILP work best for feature extraction, drastically reducing the MAE of the produced images.

Test D: We consider a fragment of the image on large images to reduce computation time. Therefore, we performed several tests to understand the impact of the selected sample on the final result. The experiments carried out are two: in the first, we examined the effect of size, considering sizes from a minimum of 20x20 to a maximum of 200x200. In the second one, we examined the effect that the choice of different portions of the image has, considering four different fragments for the identification of the filter. The features considered are illustrated in fig. 2.20. The results show that the LP model is both sample and size-dependent, while these choices do not affect the ILP model. It successfully identifies the applied filters with computational times that do not exceed 0.30 seconds in all cases.

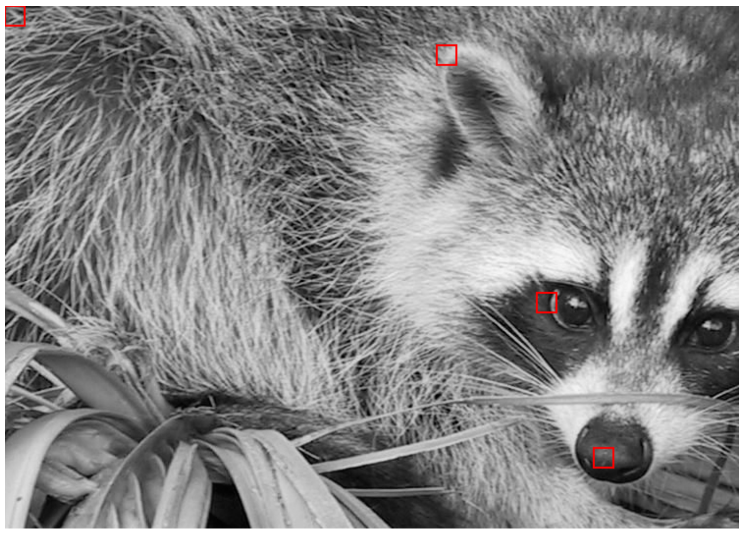


Figure 2.20: The features considered for the test D. These feature are: *TopLeft* ((0,0)-(19,19)), *Nose* ((600,450)-(619,469)), *Eye* ((550,300)-(569,319)), and *Ear* ((450,50)-(469,69)). [Capobianco et al., 2021]

Test E: Finally, in this test, we examined the behavior of the models in the case in which we want to identify a filter with different characteristics from the original. Specifically, we conducted tests looking for filters f' of size different from f (different $k \times h$), and filters considering different ranges $[-\delta, \delta]$.The results show that ILP's model is filter-size dependent and range dependent: for the former, if the filter to be identified has a magnitude greater than the selected magnitude, then the model reaches the time limit, identifying filters that produce images with $MAE \leq 12\%$, instead, selecting a size greater than or equal to the original, ILP identifies the applied filter correctly. For the latter, let $Max(f)$ be the element of f with the absolute maximum value, the closer δ is to $Max(f)$, the smaller the MAE between images. Moreover, the times follow a decreasing trend for $\delta \leq Max(f)$, until they become constant for $\delta \geq Max(f)$.

From the results, we can say that our formulations, especially the ILP one, identify a wide range of filters applicable on the input image, even if the final image is affected by noise, and with a simple 20x20 fragment of the image. In addition, our approaches can identify filters for emphasizing manually highlighted features. Finally, having formulated the problem mathematically, our approaches certify the presence of a $k \times h$ filter capable of emphasizing the selected feature. Another fundamental characteristic is their low computational complexity, which makes them usable in contexts where energy consumption must be minimized, as in our case.

2.6 Conclusions

The close enough traveling salesman problemis a generalization of the classical traveling salesman problem. The goal is to determine the shortest route that traverses all the neighborhoods of a given set of targets. This problem can model several real-world application scenarios such as meter reading. Moreover, the application potential of this problem may grow soon, given the introduction of drones.

This dissertation proposes a genetic algorithm for solving CETSP, supported by two local searches and a mathematical model. The results show that GA outperforms all other approaches in the literature, providing 32 new best solutions out of 62 instances and 27 solutions already found by others. In addition, we present two new metrics regarding instance characteristics to identify features of the problem that condition the difficulty and the structure of the solution. We performed an extended analysis to show that these new metrics provide additional information to the overlap ratio, a metric already found in the literature.

We presented a new real-world application scenario of CETSP related to solar panel diagnostics using drones. The problem is identifying the route that the drone should follow to perform a reconnaissance flight over the PV array. The application is related to a research project involving the University of Molise. With the application of the genetic algorithm presented in the previous sections, we show how we obtain considerable savings on the drone route, a key factor in the correct detection of solar panel performance.

The above work is collected in [Di Placido et al., 2021], which has been submitted to and accepted by the journal Computers and Operations Research.

Finally, still within the research project, we present three mathematical formulations for the Filter Retrieval Problem: two concerning the deconvolution process and one concerning thresholding. We formulated a linear programming (LP) and integer linear programming (ILP) model to identify deconvolution filters and performed several tests to prove their validity and performance. We can see from the results that our formulations, especially the ILP one, identify several filters applied on the input images, even if the output images are affected by noise, with a small 20x20 sample of the total image. The output images produced by the filters produced by the models have a low MAE by considering the original image as an oracle. In addition, our models can bring out features with a single manually produced image. Finally, if our approach cannot identify a filter that reliably accentuates certain features, we can argue that it is impossible to bring out those features using filters.

The results of this second part of the project were collected in the paper [Capobianco et al., 2021] and published in the journal Soft Computing.

Chapter 3

Mixed Constrained Generalized Routing Problem

3.1 Introduction

Real-world application scenarios of CETSP and CEARP have been studied in recent years due to the spread of radio-controlled or autonomous vehicles that can fly (e.g., drones). The most common case is meter reading through RFID systems [Shuttleworth et al., 2008], but it has been shown how these problems are adaptable in different contexts [Poikonen et al., 2017, Yuan et al., 2007, Di Placido et al., 2021]. Nevertheless, in a real-world scenario, the drone cannot always fly freely (CETSP) or follow the road network (CEARP). Suppose that our drone needs to fly over a school, a hospital, or a private residential area. Moreover, in the case of military or high-security zones, the crossing is forbidden. This chapter introduces the mixed constrained generalized routing problem (MCGRP), a generalization of both CETSP and CEARP. Our medium has a different degree of freedom depending on its zone. With MCGRP, we can model a mixed situation between CETSP and CEARP by combining them. We introduce the concept of flight zones associated with different flight rules. We formally present the mixed constrained generalized routing problem, a generalization, and a combination of CETSP and CEARP to model real air routing situations. Also, we provide a series of steps to follow when defining benchmark instances for the problem. Finally, we propose a phased approach to solve the problem as one of its parent problems, namely CETSP or CEARP. The chapter is structured as follows: in the section 3.2 we formally introduce the problem and the concept of flight zone. In the 3.3 section, we describe the approach used to relate the problem to others already known. In the 3.4 section we show the computational results.

3.2 Problem Definition

To define the mixed constrained generalized routing problem (MCGRP), we introduce the concept of flight zones. We differentiate between two of them: one in which it is possible to fly freely, called free-flight zone (FFZ), and a second one in which the possibility of flight is restricted to specific corridors (e.g., roads) or prohibited, called constrained flight zones (CFZ). Let N be the set of nodes on the plane, and let p_0 be the starting point named *depot*. We refer to the elements of N as targets. With each of them $v_i \in N$ is associated an area or neighborhood $N(v_i)$ of circular shape, with center in v_i and radius r_{v_i}. The goal of the mixed constrained generalized routing problem (MCGRP) is to find the shortest tour $T = \{p_0, p_i, ..., p_k, p_0\}$, $k = |N|$, which starts in depot p_0, traverses all disks $N(v_i)$, and ends in p_0.

The problem is defined on a graph $G = (N, A, V, Z)$, where N is the set of targets, A is the set of arcs, V is the set of auxiliary nodes (e.g. crossing between streets), and Z is the set of CFZs. The points p_i, $i \neq 0$ visited by the tour T are called *turning points*. Each turning point p_i is associated with a target v_i, i.e., $d(p_i, v_i) \leq r_i$, where $d(p_i, v_i)$ is the Euclidean distance between p_i and v_i, This means that a turning point p_i is the point where the tour intersects the neighborhood of the target v_i. For any target $v_i \in N$ belonging to the FFZs, the turning point p_i can be unique or any point in the segment that through $N(v_i)$. Instead, for each target $v_i \in N$ belonging to CFZs, let $a_i \in A$ be an arc through $N(v_i)$, the turning point p_i corresponds to one of the intersection points between a_i and $N(v_i)$, or any point of a_i contained in $N(v_i)$.

Given a pair of turning points p_i and p_j, the distance between them is given by the Euclidean distance $d(p_i, p_j)$ plus the distance of the arcs that make up the spline $\overline{p_i p_j}$. Hence, the total cost of the tour T is equal to $\sum_{n=1}^{|N|-1} d(p_i, p_{i+1}) + \overline{p_i p_j}$. Note that if $p_i, p_j \in FFZ$, $\overline{p_i p_j} = 0$, whereas if $p_i, p_j \in CFZ$, $d(p_i, p_j) = 0$.

3.3 Methodology

This section will present the techniques used to solve the MCGRP. Specifically, we have defined a pipeline of operations to reduce this problem to an already known one to use existing algorithms to solve it.

3.3.1 Conversion Algorithm

it is possible to trace the problem back to one already known in the literature: MCGRP with only CFZs is reducible to a CEARP; similarly, an instance of MCGRP with only FFZs can be traced back to a CETSP. From this, we can make the problem solvable as a CETSP or CEARP through a series of graph reduction transformations. In the present case, we have made the problem a CETSP so that it is solvable with the GA presented in 2.3. The following paragraphs describe the steps to solve the problem as a CETSP.

Graph Reduction: The first step is to reduce the complexity of the instance through some observations. Since the MCGRP can be traced to CETSP, we can carry over the considerations made in the section 2.3.1 for this problem as well. In addition, we can remove arcs and nodes within the FFZs. Trivially, nodes and arcs outside the CFZs are not needed since we can move freely in space. If arcs inside CFZs continue to FFZs, these are split so that only those inside CFZs are considered. Finally, we can model the perimeter of CFZs with arcs so that we also have tours that follow the perimeter without entering the zone directly. An explanatory example is shown in the figure 3.1

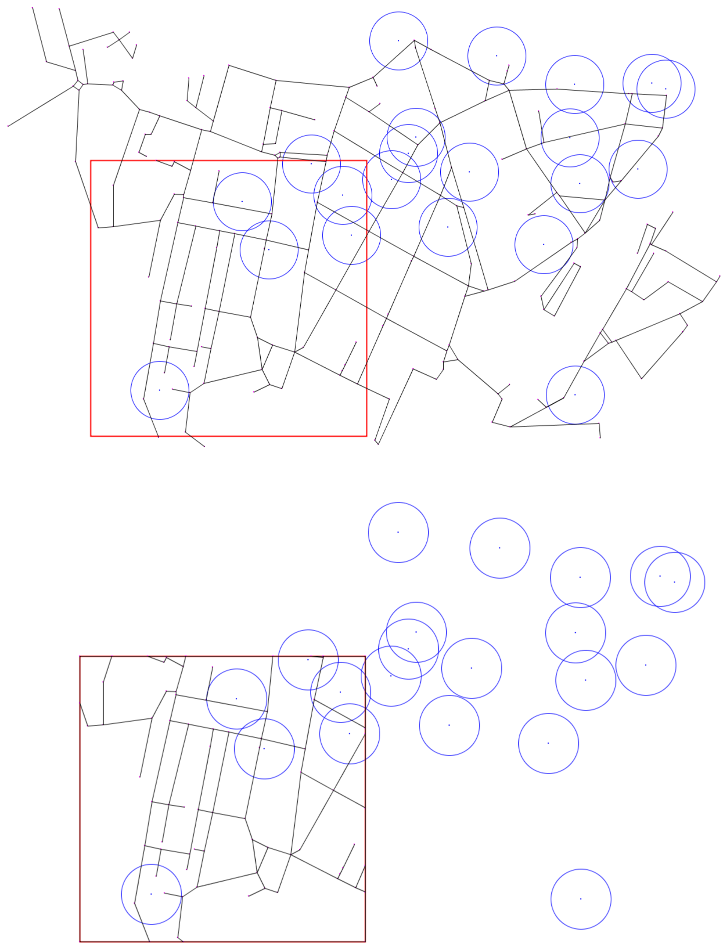


Figure 3.1: Example of graph reduction. On the top, we have the initial instance, while on the bottom we have the reduced instance.

Targets discretization: The second step consists in discretizing the targets into point sets. Given the presence of two different zones, the targets are differently discretized if they belong to an FFZs or a CFZs. For the former, we used the discretization models described in [Carrabs et al., 2017b, Carrabs et al., 2017a], where a k number of points are identified on the circumference of each target as follows. Given k, we decide on a random angle ang. We place the first discretization point; then, to obtain the remaining points, we iteratively compute ang such that k points are equidistant from each other. While for the second, we use the intersection points between the arcs and the neighborhoods of the targets. An example is shown in figure 3.2.

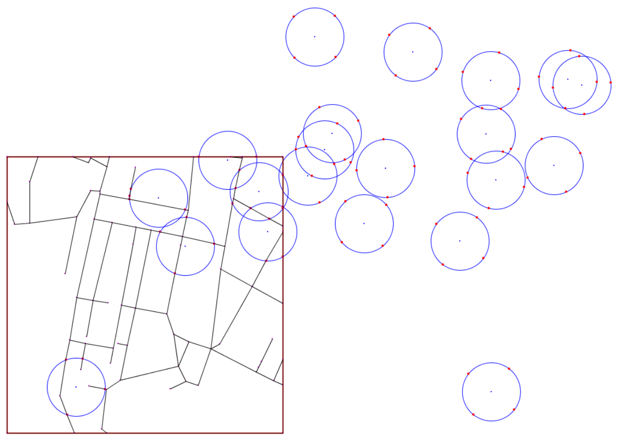


Figure 3.2: Example of targets discretization

Arches generation: After defining the discretization points for each target, we need to connect them through arcs. To do this, we consider the discretization points in pairs and generate an arc between them. The points belong to different targets since, trivially, it is not necessary to connect points that discretize the same target. The second set of generated arcs connects the discretization points with the intersection points of the CFZs to the FFZs. Finally, an additional set of arcs joins the intersection points of the CFZs to each other to connect the various CFZs. Figure 3.3 shows the result of creating the arcs on an instance.

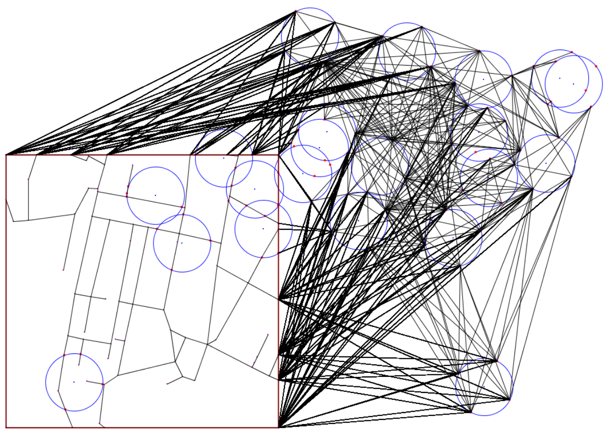


Figure 3.3: Result of the arches generation step

Graph generalization: After creating the additional arcs, the problem is solvable as a CEARP. In fact, the end result of the previous steps is a graph $G' = (N, A', V)$, where A' contains all the arcs of A with the addition of the newly generated arcs. On the contrary, an additional step is required to lead it back to a CETSP. Starting from G' we define the complete graph $G'' = (V'', A'')$, where V'' is the set of nodes corresponding to the targets of the graph G, and A'' is the set of arcs connecting the nodes. Specifically, let $v''_i \in V''$ a generic node of graph G'', a generic arc $a'' = (v''_i, v''_j) \in A''$ is a path defined by a subset of arcs $\{a'_i, a'_j, ..., a'_k\} \subseteq A'$ such that the starting point is v''_i and the ending point is v''_j. The set A'' is easily obtained by solving a shortest path problem on the graph G' for each pair of nodes $v_i, v_j \in V$. At this point, G'' configures a GTSP and can be solved through algorithms known in the literature, such as the one presented in [Carrabs et al., 2017b] for solving CETSP.

3.4 Computational Results

Several tests have been conducted to verify the validity of our methods. In the context of this work, we mainly focused on the study of the limiting case containing only FFZs, then on CETSP. A genetic algorithm, presented in section 2.3, was implemented and tested on the instances of the literature for CETSP. For more information on this, refer to section 2.5.

3.5 Conclusions

CETSP and CEARP have been extensively studied in recent years for their ability to model a large number of real-world applications, especially related to drone use. The most common case is meter reading through RFID systems [Shuttleworth et al., 2008], but it has been shown how these problems are adaptable in different contexts [Poikonen et al., 2017, Yuan et al., 2007, Di Placido et al., 2021]. Nevertheless, in a real-world scenario, the drone cannot always fly freely (CETSP) or follow the road network (CEARP). Suppose that our drone needs to fly over a school, a hospital, or a private residential area. Moreover, in the case of military or high-security zones, the crossing is forbidden. We presented a mixed constrained generalized routing problem generalization of both CETSP and CEARP problems. Our medium has different degrees of freedom depending on the area it is traversing. This generalization can model a mixed situation where both problems are present together. We introduced the concept of flight zones, where different flight rules are associated with each of them. We have presented a simple scheme to follow to define benchmark instances, and, further, we have defined a pipeline of operations to relate the above problem to the parent ones. As part of this work, we traced the original problem back to a CETSP and tested our approach, presented in the 2.3 section.

Chapter 4

Generalized Close Enough Traveling Salesman Problem

4.1 Introduction

The introduction of drones as a means of freight transportation opens up research to various applications in logistics and reconnaissance. The generalization of the classical traveling salesman problem, presented as the close enough traveling salesman problem, has attracted researchers in recent years because of its ability to model most of the real-world applications possible with drones. We recall that the CETSP perfectly describes all those applications. It is necessary to determine the shortest tour that starts at a specific point, called *depot*, crosses a set of areas associated with the customers to be covered, generally circular returns to the depot. In the literature, we find several examples of real-world problems modeled through CETSP [Shuttleworth et al., 2008, Poikonen et al., 2017, Yuan et al., 2007, Di Placido et al., 2021], and we find as many solving methods, ranging from exact algorithms [Behdani and Smith, 2014, Carrabs et al., 2017b, Carrabs et al., 2017a, Coutinho et al., 2016] to heuristics and metaheuristics that can provide reasonable upper and lower bounds for the problem. Furthermore, metrics for examining the instance complexity from its morphology and resolution have also been defined and studied [Yang et al., 2018, Wang et al., 2019, Carrabs et al., 2020, Di Placido et al., 2021].

This chapter presents a generalization of CETSP called the generalized close enough traveling salesman problem (GCETSP). Each customer has several action areas (neighborhoods), instead of only one, all centered on its location with radii that differ from each other. To each of these areas is associated a prize collected to the passage of that region. Introducing different neighborhoods for each customer allows modeling of different real-world application contexts, in which we get greater rewards if we approach close to the targets. For example, when considering RFID meter reading systems, the rewards can represent the probability of successful meter reading. This probability decreases as we move away from the customer, which explains the decreasing reward with respect to the length of the disk radius. The goal of the GCETSP is to determine an allowable route that maximizes the difference between the total collected reward and the route length. By an eligible route for the problem, we mean a route that visits exactly one disk per customer and depot.

This part of the dissertation is structured as follows. The problem and the notations used are described in the 4.2 section, while the 4.3 section is devoted to the solving approaches. In the section 4.4 we present instance generation, computational results, and additional analysis.

4.2 Problem definition and notations

Let N be the set of customers arranged on a Euclidean plane, and let n_0 be the starting point named *depot*. For each customer $n_i \in N \backslash n_0$, we define a set of concentric disks K_i in n_i of varying radius, i.e., $r^j_{n_i} < r^{j+1}_{n_i}, j+1, ..., |K|-1$, where r^j_c is the radius of the disk K_j with center in n_i. Each disk K_j is associated with a premium p^j_c that grows as the radius decreases, i.e., $p^j_c > p^{j+1}_c, j+1, ..., |K|-1$. The depot is associated with a circle of radius 0 with prize 0. The goal of GCETSP is to determine the tour that maximizes the difference between the total prize collected and the length of the tour, and that visits at most one circle per customer.

We refer to the points visited by tour T that traverse a single disk associated with customer n_i as *turning point* tp_i. Unlike CETSP, in this problem we define as a generic turning point tp_i a triple (x_i, y_i, r^t_i) associated with customer $n_i \in N$, where x_i, y_i are the coordinates of the point on the Euclidean plane such that $d(tp_i, n_i) \leq r^t_i$, where $d(tp_i, n_i)$ is the Euclidean distance between points tp_i and n_i. This means that tp_i is that point where the tour crosses customer n_i on the circumference K^t_i. An explanatory example is shown in figure 4.1.

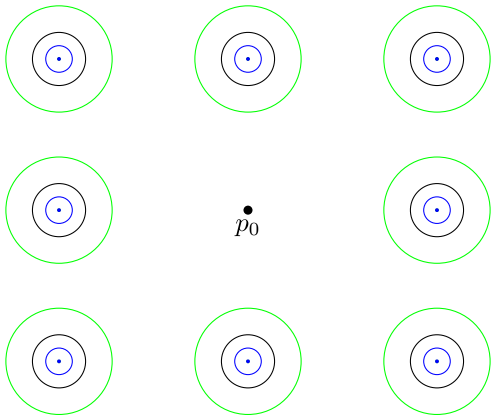


Figure 4.1: Example of GCETSP instance. The point p_0 is the depot. Each target has three concentric disks categorized as follows: the green ones are the outer, the black ones the mids, and the blue ones the inners.

4.3 Methodology

In this section we describe two solving heuristics for solving GCETSP. The first is the genetic algorithm (GA), described in 4.3.1, while the second is a constructive algorithm called nearOpt, described in 4.3.2.

4.3.1 Genetic algorithm

The genetic algorithm we propose below is an adaptation of the one presented in 2.3 to solve CETSP. Given the similarity between the two problems, we adapted our approach for CETSP to this new variant. Below, we provide a brief description of the GA highlighting the changes applied to handle GCETSP.

Chromosome encoding and fitness function The chromosome is a representation of the solution for GCETSP. Each chromosome is an ordered sequence of genes, each representing a *turning point* in the tour they describe, as shown in figure 4.2

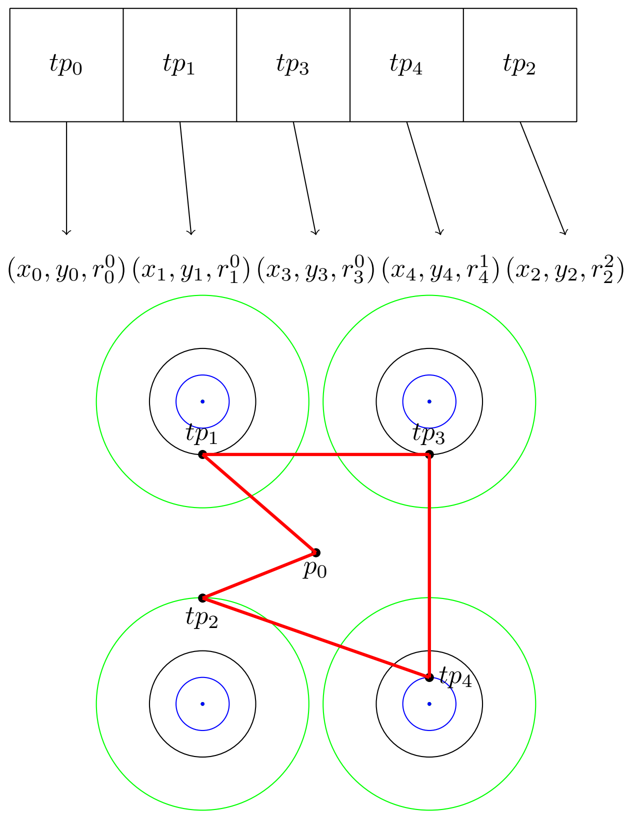


Figure 4.2: Example of chromosome with the relative turning points. Each target c_i has three concentric disks: mid (black circle), inner (blue circle), and outer (green circle), with different radii r_i^0, r_i^1 and r_i^2, respectively. The tour derived by the chromosome is highlighted in red.

As mentioned above, the turning points also have information about the disk on which they are placed, since each customer has more than one disk from which we can choose.

Each chromosome is evaluated through a fitness function. In the case of GCETSP, this function is the difference between the collected reward $P(T)$ and the total length of the tour $w(T)$ corresponding to the sequence of turning points T.

$$f(T) = P(T) - w(T)$$

Initial population and generation of new individuals The population of each generation has a fixed number of chromosomes, pop_{size}. We generated the population randomly, as in the first version of the approach. The significant difference is the random choice of disks on which the turning points are placed.

Specifically, for each customer $n_i \in N - \{n_0\}$, we randomly choose a disk K_i^t and generate a turning point that crosses it. The location is chosen at random on the circumference. Finally, we obtain an admissible tour T by considering a random permutation of these turning points with the addition of the depot at the beginning and end of the tour. This process is repeated until the population is filled.

The population of each generation is obtained in the same way as the previous version of the GA. We define two population halves by two steps: Let $P = \{c_0, c_1, ..., c_k\}$ be the current population, where $c_i \in P$ is a generic chromosome, we sort the elements of P in descending order by fitness value. From P we extract the first $\frac{|P|}{2}$ to be preserved for the next generation P'. Finally, to fill P', we generate the second part by applying the crossover operator described in 2.3.5.

In addition, we use a mutation operator to avoid premature convergence of the algorithm and preserve heterogeneity in the population. The operator is applied with a certain probability on every chromosome in the first half of the population. The value of this probability is equal to the number of generations without improvement divided by 100. The operator chooses an arbitrary interval $[i_1, i_2]$, with i_1, i_2 computed randomly. Then, each gene belonging to the interval is replaced with another random one. Unlike CETSP, in GCETSP, we have multiple neighborhoods for each customer. For this, the position of the new turning point is determined on a randomly chosen K_j^k disk.

Improvement procedures Enhancement procedures are used when a chromosome is created or modified to improve its fitness value. For CETSP, we implemented three procedures: a 2opt, intending to improve the crossing sequence of the turning points without changing their position, and a second-order cone programming algorithm (SOCP) and a bisection algorithm (3Alg), to optimize the position of the turning points by fixing the visiting sequence. We remember that the bisection algorithm has been introduced to lighten computation times. Although the SOCP supplies the optimal position of the turning points, if executed repeatedly, it burdens the execution of the GA, even if the resolution happens in polynomial time. In contrast, the bisection algorithm supplies a heuristic solution to the problem of determining the optimal position of the turning points.

In this variant, in addition to the three procedures described above, we propose a fourth, called improveSolution, to improve the rewards collected by modifying the disks on which the turning points are placed. This procedure works as follows.

Starting from a tour T, improveSolution considers all consecutive triplets of turning points tp_i, tp_j, tp_k of T. For each triplet, the extreme points are fixed, while the intermediate point is the one to be repositioned. The goal is to identify the disk K_j^t of target n_t such that the difference between the reward collected in crossing the disk and the length of the triplet is maximized. Let r_j^k be the radius of the current disk currently associated with tp_j, we compute $tp_j^{'} = (x_j^{'}, {y_j}', r_j^t)$

s.t. $r_j^t \neq r_j^k$. This point is identified through the bisection algorithm introduced above and described in section 2.3.7. Then, if $p_j^t - d(tp_i tp_j^{'}) + d(tp_j^{'} tp_k) \geq p_j^k - d(tp_i tp_j) + d(tp_j tp_k)$, then $tp_j^{'}$ replaces the old turning point of the target n_j. This step is repeated $\forall r_j^t \neq r_j^k$ associated with n_j. The procedure ends when all triplets are considered. In case a point is replaced by another one on a different disk, the procedure resumes from its predecessor, since it may be convenient to move the previous point as well. An example of how improveSolution works is shown in Figure 4.3

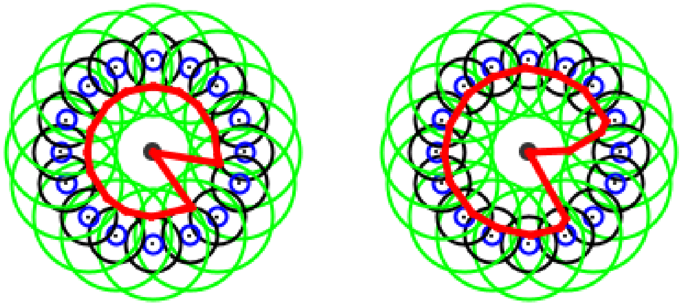


Figure 4.3: Example of improveSolution application. On the left we have the original tour, while on the right the improved one.

Stopping criteria The genetic algorithm terminates when it reaches it_{size} iterations or when there is no improvement on the best fitness value after a specific number of generations it_{impr}. At the end of execution, the best chromosome in the current population is returned as output.

4.3.2 nearOpt

nearOpt is a constructive algorithm that works as follows. Initially, it identifies the traversal sequence of the customers, e.g., by solving a TSP considering only the location of the customers. Once the sequence is known, we only need to identify which disks to traverse for each customer. Then, for each customer $n_i \in N - n_0$, we produce $|K_i|$ points, one for each disk. The algorithm uses SOCP by considering one disk at a time. For example, if we consider $K_i^0, \forall n_i \in N - \{n_0\}$, using SOCP the result is a set of turning points T^0, positioned at the optimum according to the considered sequence, such that $\overline{tp_i n_i} \leq r_i^0, i = 1, ..., |N|$. nearOpt solves the shortest path problem on these points to obtain a solution for GCETSP. Let $Q_i = \{tp_i^0, tp_i^1, ..., tp_i^{|K_i|}\}$ be the set of turning points on the neighborhoods relative to customer n_i. We construct a graph where the first and last nodes correspond to the depot n_0. Then, the $|K_i|$ points associated with customer n_i are connected only with the predecessor and successor of the sequence. For each of these, we have an incoming edge from each predecessor's points and an outgoing edge to each successor's point. Trivially, the first node has no incoming edges, while the last node has no outgoing edges. Unlike the classical shortest path problem, the weight of the edges also includes the rewards related to the disks. Specifically, since each point is located on a specific disk, it reaps the reward associated with that disk if crossed. From this, the weight of each arc is calculated as:

$$w(e) = \overline{tp_i^k tp_{i+1}^j} - p_{i+1}^j$$

Where tp_i^k is the point associated with the customer n_i placed on the K_i^k disk, tp_{i+1}^j is the point associated with the successor n_{i+1} placed on the K_{i+1}^j disk, and p_{i+1}^j is the collectible prize relative to the K_{i+1}^j disk. For the first and last nodes, there is only one point with associated edges having prize equal to 0. An example of the graph produced is shown in figure 4.4.

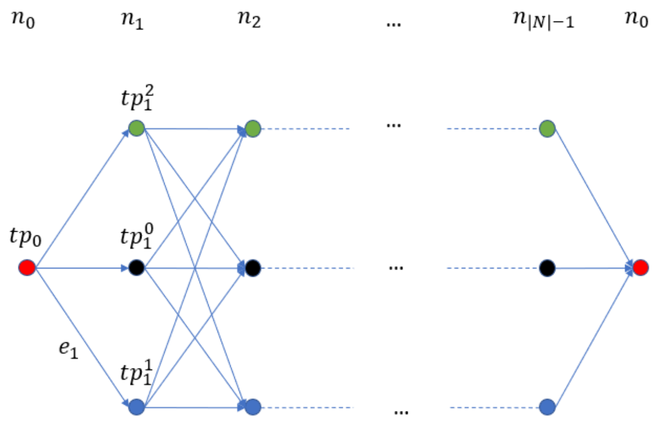


Figure 4.4: Example of the graph associated with the sequence $\{n_0, n_1, n_2, ..., n_{|N|-1}, n_0\}$ on an instance with three disks for each target. The red points are the depot. The green points are related to the outer disks, the black ones to the mid disks, and the blue ones to the inner disks. The points associated with a customer are connected only with the points of the successor. The edges' weights are calculated as the difference of the points' distance and the prize collected by traversing the destination point. For instance, the weight of e_1 is equal to $\overline{tp_0 tp_1^1} - p_1^1$.

The solution of the shortest path problem determines which disks are traversed for the various customers so that the maximum gain is obtained while minimizing the tour length at the same time. Finally, we apply a SOCP to improve the turning points' location further.

4.4 Computational tests

We devote this section to the results and analysis of the experiments performed. Given the novelty of the problem, the first part is devoted to the description of instance generation for GCETSP. Then, a second section emphasizes the differences between the original CETSP problem and its GCETSP variant through the conformation of the solutions. Finally, we evaluated the performance of the GA and nearOpt presented in the 4.3 section. The experiments were conducted on a machine equipped with Windows 10 Home as the OS, an AMD Ryzen 7 3750H 2.30 GHz processor, and 16 GB of RAM, and both algorithms were written in the Java language (JDK 13.0). The GA parameters were set as in the tests performed for CETSP: the number of individuals per population pop_{size} was set to 50, the number of generations it_{size} equal to 1000, and it_{impr} equal to $\frac{it_{size}}{20}$.

4.4.1 Instance generation

GCETSP is a new variant of the famous CETSP and needs a set of benchmarks to test the solving approaches. Given the similarity to its main problem, it is possible to use the instances proposed for CETSP in [Mennell, 2009] and adapt them for GCETSP. It was possible by adding two disks together with the one already present in the CETSP instances. Let r_i^0 be the radius of the original disk K_i^0 a of generic customer $n_i \in N - \{n_0\}$; the inner disk K_i^1 has radius $r_i^1 = \frac{r_i^0}{3}$, and the outer disk K_i^2 has radius $r_i^2 = 2r_i^0$.

We classified the instances following the logic presented in the section 2.5. *Varied overlap ratios* includes the instances with a fixed radius for all customers but with different overlap ratios between them. *Overlap Ratio* contains the instances whose radii have been adjusted to have a specific OR value, and in *Arbitrary Radius* we find all the instances for which the customers have different radii.

In addition to these instances, we propose a second set of instances, called *convex hull* instances, for which the optimal solution is easily obtained or deduced. These are produced as follows: given a set of points randomly arranged on the plane, we determine the smallest convex polygon containing them (convex hull). The polygon's vertices will make up the set of N customers, while the remaining points are discarded. Finally, we add the depot $n_0 = (0, 0)$ to the set. For each customer except the depot, we define three disks: $\forall n_i \in N - \{n_0\}$, $r_i^0 = 10, r_i^1 = \frac{r_i^0}{3}, r_i^2 = 2r_i^0$. Figure 4.5 shows an example of a convex hull instance.

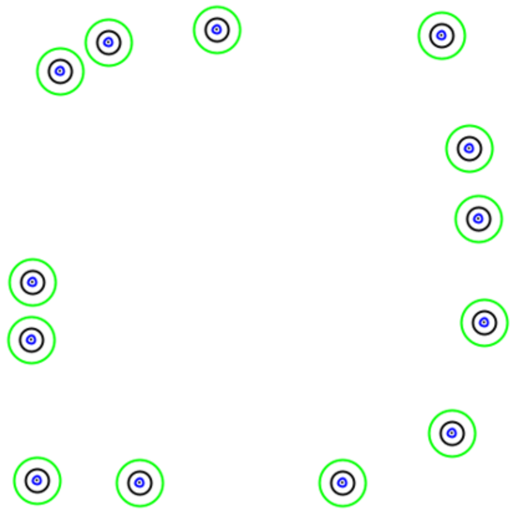


Figure 4.5: Example of an instance produced by the convex hull method. The green circles are the outer disks, the black ones the mid disks, and the blue ones the inner disks.

As for the rewards, we propose a simple strategy based on the ratios of the radii (RR). Let k be the step and r_i^x the radius of the disk K_i^x, the reward relative to the generic disk K_i^x is equal to $p_i^t = \frac{r_i^t}{r_i^x} \times k$. For convenience, in our experiments $r_i^x = r_i^0$ is the radius of the original disk K_i^0 of the customers. For example, considering the above instances, the ratios are $\frac{r_i^0}{r_i^0} = 1$, $\frac{r_i^1}{r_i^0} = 3$ and $\frac{r_i^2}{r_i^0} = \frac{1}{2}$. Therefore, with $k = 2$, we get $p_i^0 = 2$, $p_i^1 = 6$ and $p_i^2 = 1$.

The rewards associated with customer neighborhoods weigh heavily on the structure of the optimal solution for obvious reasons: the greater the gain obtainable from internal disks, the greater the chance of going inward since this deviation has little impact on the objective function. Recall that the problem's objective function is the difference between the prizes collected and the length of the tour. On the contrary, external disks are preferred if the gain is low since deviating inside is inconvenient. Based on this, we performed several tests to identify the right pricing configuration for the instances described earlier to maximize balance and heterogeneity in disk selection. We tried four different combinations of rewards computed with the RR strategy: $k = 1, 2, 3, 5$. The results are reported in the table 4.1 where, for each k, we record the percentage of disks traversed by the GA solutions, divided into *mid*, *inner* and *outer*.

We can see that we prefer internal disks for high values of k. Instead, with low values of k, we prefer external disks. Nevertheless, a suitable configuration for

k=1			**k=2**			**k=3**			**k=5**		
mid	*inner*	*outer*	*mid*	*inner*	*outer*	*mid*	*inner*	*outer*	*mid*	*inner*	*outer*
9.76%	81.07%	9.15%	4.96%	90.56%	4.48%	3.00%	94.86%	2.13%	0.69%	98.75%	0.56%

Table 4.1: Disks traversed by GA solutions on adapted CETSP instances considering different prizes using RR strategy.

GCETSP instances turns out to be when $k = 1$, which provides the best variance of disk selection. So, we consider instances with this pricing configuration in the remainder of this section.

4.4.2 CETSP comparison

An initial set of tests was performed to compare CETSP and GCETSP solutions. The purpose is to analyze the differences between the two problems and examine how much their respective solutions differ from each other. We considered adapted CETSP instances for these experiments. The comparison was made between three CETSP solutions and the GCETSP solution computed through the GA presented in 4.3. The three CETSP solutions were obtained by solving a CETSP using the approach described in 2.3 considering only one type of disk at a time, e.g., external only, intermediate only, or internal only. The results are shown in the tables 4.2, 4.3, and 4.4. These tables have the same grouping of instances proposed in the previous CETSP results. They are organized as follows: the first column shows the name of the instances, while the next three sets of columns display the values relative to CETSP solutions considering only intermediate, internal, or external disks at a time, respectively. For each of these, we report the value of the solutions, the computational time in seconds, and the percentage gap from the GCETSP solution, calculated through the formula: $\frac{GA_{value}-value}{GA_{value}}$, where GA_{value} is the value of the GCETSP solution. At the same time, $value$ is the value of the CETSP solution on the corresponding disk. The best solution values are presented in bold. Finally, the last column contains the GA solutions, where we also list the disks selected from the solution. It is important to note that several values in the *value* columns are negative due to the differences between premium and length. For this reason, the percentage gaps are reported as absolute values, and for instances where the GA performs worse than the CETSP solutions, we report it as negative. We decided to use this notation to avoid ambiguity in reading.

Instances	**mCETSP**			**iCETSP**			**oCETSP**			**GA**				
	value	*time*	*gap*	*value*	*time*	*gap*	*value*	*time*	*gap*	*value*	*time*	*mid*	*inner*	*outer*
bonus1000	588.15	8786.91	69.57%	**2142.02**	23034.32	-10.82%	267.15	7580.49	86.18%	1932.93	13886.54	142	824	34
bubbles1	-313.13	2.21	25.81%	-295.10	1.20	18.56%	-253.67	2.41	1.92%	**-248.90**	1.32	4	19	13
bubbles2	-352.28	2.22	45.75%	-321.65	2.57	33.08%	-310.95	1.75	28.65%	**-241.70**	6.76	24	31	21
bubbles3	-403.96	7.29	50.08%	-567.37	21.14	110.79%	-384.28	3.24	42.77%	**-269.16**	22.80	56	40	30
bubbles4	-621.46	27.84	101.58%	-667.35	31.44	116.47%	-454.16	12.95	47.31%	**-308.29**	56.41	60	54	70
bubbles5	-788.16	45.07	71.81%	-981.79	100.41	114.02%	-526.78	35.97	14.83%	**-458.73**	88.71	65	88	97
bubbles6	-905.66	176.16	61.63%	-1204.14	170.77	114.89%	-711.14	155.01	26.91%	**-560.34**	257.61	89	109	126
bubbles7	-1501.31	809.15	141.66%	-2423.71	920.56	290.14%	-880.97	254.67	41.81%	**-621.24**	443.75	123	130	153
bubbles8	-1750.72	1230.27	114.04%	-2738.59	1976.35	234.81%	-1038.14	1136.83	26.92%	**-817.96**	776.32	146	169	181
bubbles9	-1665.22	2655.50	81.32%	-2216.79	3587.61	141.38%	-1158.51	1107.48	26.14%	**-918.40**	1145.50	153	208	233
chaoSingleDep	-839.61	10.69	53.36%	-551.76	39.33	0.78%	-814.19	14.11	48.72%	**-547.47**	24.28	25	175	0
concentricCircles1	-37.16	0.94	101.06%	-21.54	0.45	16.57%	-21.63	0.50	17.03%	**-18.48**	0.47	1	12	3
concentricCircles2	-117.13	2.77	74.02%	-83.56	0.85	24.14%	-87.57	0.83	30.10%	**-67.31**	1.33	1	23	12
concentricCircles3	-210.04	2.79	60.12%	-172.16	1.82	31.25%	-168.03	1.39	28.10%	**-131.17**	4.47	1	35	24
concentricCircles4	-348.64	12.56	65.53%	-264.81	9.46	25.73%	-276.01	6.38	31.05%	**-210.62**	14.61	9	61	34
concentricCircles5	-484.99	26.04	62.37%	-376.77	20.77	26.14%	-400.15	8.04	33.97%	**-298.69**	18.03	8	86	54
rotatingDiamonds1	-12.29	0.80	171.41%	**17.21**	0.39	0.00%	-8.59	0.38	149.92%	**17.21**	0.67	0	20	0
rotatingDiamonds2	-80.48	1.94	527.12%	**18.84**	0.96	0.00%	-80.14	2.02	525.31%	**18.84**	1.00	0	60	0
rotatingDiamonds3	-200.88	10.60	257.73%	**127.36**	8.38	0.00%	-245.70	41.66	292.92%	**127.36**	6.57	0	180	0
rotatingDiamonds4	-450.66	41.85	408.16%	**146.24**	32.74	0.00%	-543.17	93.51	471.41%	**146.24**	24.41	0	320	0
rotatingDiamonds5	-830.75	376.41	274.97%	**474.79**	484.20	0.00%	-1089.69	782.30	329.51%	**474.79**	159.44	0	680	0
team1_100	-207.34	3.86	200.38%	-138.53	4.16	100.70%	-173.67	2.05	151.61%	**-69.02**	9.63	19	72	9
team2_200	-46.68	23.75	132.50%	99.01	25.60	31.06%	-38.83	2.82	127.04%	**143.62**	44.87	71	116	10
team3_300	-164.20	99.30	153.37%	198.60	117.86	35.45%	-174.22	44.40	156.63%	**307.65**	136.02	22	243	34
team4_400	-285.52	428.43	206.31%	177.78	812.95	33.81%	-256.38	285.82	195.46%	**268.57**	441.42	31	325	44
team5_499	-201.50	995.23	132.80%	582.72	1319.79	5.14%	-304.41	847.66	149.55%	**614.30**	384.72	2	468	29
team6_500	274.78	518.12	71.95%	**1046.83**	693.98	-6.87%	117.67	43.41	87.99%	979.56	2106.47	58	442	0

Table 4.2: Comparison between CETSP and GCETSP solutions on CETSP adapted instances with varied overlap ratios. The CETSP solutions are computed considering one disk at a time: mCETSP on mid disks, iCETSP on inner disks, and oCETSP on outer disks.

Instances	**mCETSP**			**iCETSP**			**oCETSP**			**GA**				
	value	*time*	*gap*	*value*	*time*	*gap*	*value*	*time*	*gap*	*value*	*time*	*mid*	*inner*	*outer*
Overlap ratio 0.02														
d493	289.77	864.47	75.95%	**1204.88**	836.70	0.00%	91.27	741.98	92.42%	**1204.88**	320.38	0	492	0
dsj1000	60.29	20325.36	96.14%	**1610.43**	13975.89	-3.18%	-157.08	9892.99	110.06%	1560.87	1726.90	21	900	78
kroD100	-60.04	9.05	157.05%	**105.23**	4.13	0.00%	-78.35	5.18	174.46%	**105.23**	3.68	0	99	0
lin318	-2521.54	174.28	39.36%	-2677.24	153.79	47.96%	-2043.76	229.54	12.95%	**-1809.38**	169.88	40	146	131
pcb442	118.46	826.50	86.86%	**901.52**	589.50	0.00%	-16.99	550.04	101.88%	**901.52**	178.69	0	441	0
rat195	35.68	55.23	90.52%	**376.20**	55.76	0.00%	-20.19	51.70	105.37%	**376.20**	26.20	0	194	0
rd400	-633.04	686.27	1707.48%	-132.16	694.92	277.33%	-578.39	844.78	1551.44%	**-35.02**	456.07	1	320	78
Overlap ratio 0.1														
d493	390.69	345.37	70.10%	**1309.83**	1063.72	-0.24%	164.14	462.08	87.44%	1306.76	614.51	3	489	0
dsj1000	622.13	3951.05	66.08%	**2269.65**	10205.80	-23.76%	247.78	2082.36	86.49%	1833.93	2135.72	198	733	68
kroD100	9.33	3.92	94.26%	160.50	5.32	1.32%	-23.04	2.08	114.17%	**162.64**	10.99	2	96	1
lin318	-1087.89	74.77	94.42%	-1432.04	129.46	155.93%	-779.78	57.23	39.36%	**-559.55**	209.08	98	148	71
pcb442	295.18	466.16	72.20%	1057.02	433.24	0.45%	121.16	352.09	88.59%	**1061.84**	566.24	4	436	1
rat195	125.86	21.31	72.40%	453.01	36.30	0.67%	41.09	7.06	90.99%	**456.06**	65.64	0	194	0
rd400	-59.41	612.93	115.67%	350.96	500.70	7.42%	-87.06	148.60	122.96%	**379.11**	717.80	84	298	17
Overlap ratio 0.3														
d493	422.24	233.54	69.07%	**1374.57**	244.35	-0.70%	204.91	266.43	84.99%	1365.03	934.31	2	490	0
dsj1000	799.05	1181.44	64.05%	**2619.68**	5646.41	-17.86%	419.50	24.52	81.13%	2222.68	2353.69	159	801	39
kroD100	40.46	2.20	80.48%	**207.29**	1.56	0.00%	18.35	0.58	91.15%	**207.29**	6.51	0	99	0
lin318	-448.96	29.69	129.82%	-450.91	24.96	130.81%	-245.92	4.81	25.88%	**-195.36**	127.35	108	194	15
pcb442	357.46	121.23	69.54%	**1178.68**	115.64	-0.43%	169.27	32.36	85.58%	1173.68	510.40	1	440	0
rat195	148.30	6.49	71.14%	**513.81**	5.58	0.00%	70.80	1.66	86.22%	**513.81**	47.05	0	194	0
rd400	174.16	67.12	75.10%	**723.06**	62.01	-3.38%	82.19	11.53	88.25%	699.41	757.13	60	336	3

Table 4.3: Comparison between CETSP and GCETSP solutions on CETSP adapted instances with fixed overlap ratio. The CETSP solutions are computed considering one disk at a time: mCETSP on mid disks, iCETSP on inner disks, and oCETSP on outer disks.

Instances	mCETSP			iCETSP			oCETSP			GA				
	value	*time*	*gap*	*value*	*time*	*gap*	*value*	*time*	*gap*	*value*	*time*	*mid*	*inner*	*outer*
bonus1000rdmRad	67.93	480.23	95.88%	**1696.89**	5463.80	-2.97%	-239.46	38.75	114.53%	1647.95	4011.67	50	902	48
d493rdmRad	357.72	5.58	72.39%	**1296.36**	19.73	-0.05%	129.80	1.39	89.98%	1295.73	27.56	0	490	2
dsj1000rdmRad	374.25	25.86	82.80%	2143.32	213.25	1.52%	-47.76	5.23	102.19%	**2176.49**	186.28	36	943	20
kroD100rdmRad	-42.83	6.97	135.82%	119.20	6.51	0.32%	-70.57	1.58	159.01%	**119.58**	8.73	0	98	1
lin318rdmRad	-1730.42	8.71	59.19%	-1744.60	50.30	60.50%	-1561.23	2.38	43.63%	**-1087.01**	55.98	34	224	59
pcb442rdmRad	221.00	16.16	78.44%	**1027.03**	155.93	-0.20%	34.03	4.04	96.68%	1024.94	108.44	1	440	0
rat195rdmRad	125.78	1.35	74.78%	497.90	1.11	0.15%	35.80	0.42	92.82%	**498.64**	31.27	1	193	0
rd400rdmRad	-847.69	701.06	307.44%	-231.93	697.39	11.48%	-866.98	254.56	316.71%	**-208.05**	235.30	0	367	32
team1_100rdmRad	-288.54	1.84	129.54%	-173.56	4.90	38.08%	-293.21	0.78	133.26%	**-125.70**	11.21	12	79	9
team2_200rdmRad	-413.66	16.04	214.45%	-231.73	36.21	76.15%	-401.65	6.80	205.32%	**-131.55**	82.33	19	146	35
team3_300rdmRad	-78.09	3.75	118.28%	357.44	12.83	16.33%	-174.42	0.84	140.83%	**427.20**	211.46	30	259	11
team4_400rdmRad	-600.31	317.04	3384.68%	-22.46	621.90	222.88%	-612.83	68.79	3453.19%	**18.28**	228.07	5	355	40
team5_499rdmRad	52.81	2.52	94.61%	942.77	21.76	3.72%	-139.56	1.43	114.25%	**979.18**	22.85	14	470	15
team6_500rdmRad	-120.98	26.15	118.57%	643.79	330.64	1.19%	-275.87	3.97	142.34%	**651.57**	144.84	39	427	34

Table 4.4: Comparison between CETSP and GCETSP solutions on CETSP adapted instances with random radii. The CETSP solutions are computed considering one disk at a time: mCETSP on mid disks, iCETSP on inner disks, and oCETSP on outer disks.

The genetic algorithm identifies solutions for GCETSP that differ particularly from CETSP solutions: as for the *mCETSP* the solution gaps ranging from a minimum of 25.81% (*bubbles1*) to a maximum of 3384.68% (*team4_400rdmRad*), while for *iCETSP*, the solution gaps spaces from a minimum of 0.15% (*rat195rdmRad*) to a maximum of 290.14% (*bubbles7*). Finally, with respect to *oCETSP*, the solution gaps varies from a minimum of 1.92% (*bubbles1*) to a maximum of 3453.19% (*team4_400*). Note also how several instances are easily solved by considering only internal disks, such as *rotatingDiamonds* instances. This scenario generally happens on instances with a large density of customers that overlap each other, as reported in the analysis presented in 2.5.2. In these cases, inner disks are much more frequent, as they do not impact the final solution too much. Nevertheless, we noticed the same behavior for instances with OR fixed at 0.02. It is reasonable since, with low ORs, the CETSP can be traced to a TSP. Again, we have that inner disks are preferred over outer disks since going towards the center has little impact on the solution. Also, it is essential to note that some instances prefer external disks, i.e., *bubbles* ones. *oCETSP* solutions are generally better than their CETSP counterparts for these instances. Finally, we can see how considering one disk at a time is a poor strategy for GCETSP resolution. A heterogeneous choice of disks is required to obtain reasonable GCETSP solutions.

Regarding the computational times, we can say that they are relatively contained, except for large instances whose resolution is heavy, such as *bonus1000* and *dsj1000*. In general, considering only external disks results in a faster resolution than their counterparts, as we can see from *oCETSP* times, while considering only internal disks results in slower resolution, as we can see from *iCETSP*. It is reasonable since considering only one type of disk corresponds to resolving a CETSP with a given OR, low or high depending on whether internal or external disks are considered, respectively. As we have shown in section 2.5.2, instances with a high OR value are computationally easier to solve. For the genetic algorithm, the resolution times are relatively short. It is important to note that when the solution of an instance contains a prevalence of external disks, such as for *bubbles*, the computational times are reduced.

4.4.3 GA performance

In the previous tests, we show how GA identifies better solutions than those based on the resolution of the classical CETSP. However, this is not sufficient to affirm the capabilities of our algorithm. To do so, we performed additional tests to compare the solutions produced by the GA with optimal or pseudo-optimal solutions. In these tests, we used the convex hull instances described in 4.4.1, for which the optimal solution is easily identifiable. For how we structured the instances in terms of customer locations, identifying the visit sequence is straightforward. So, it only remains to identify which disk to traverse for each customer. To do this, we assume that nearOpt can do this at the optimum for these instances. The algorithm works by taking a visit sequence as input and extensively exploring the search space to find the best location of the turning points. For instances where we can quickly identify the optimal sequence, we assume that we will produce solutions that tend towards optimality. For convex hull instances, the optimal visit sequence corresponds to the convex hull with the addition of the depot at the beginning and end. Based on this, we use nearOpt to produce solutions to estimate optimal solutions from a given sequence. In the figure 4.6 we show a solution obtained from the latter.

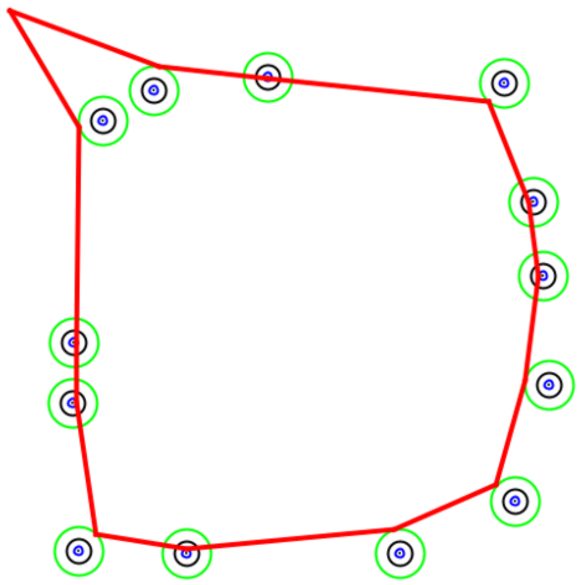


Figure 4.6: Example of a solution for the instance produced by nearOpt.

We cannot say with certainty that the solutions produced by nearOpt are great, but we can say that if they are not, they are close to being so.

We compared the nearOpt approach with GA, and the results are reported in the table 4.5 organized as follows: In the first column, we report the instance name, while in the following two columns, we provide the solution values and computational times of nearOpt and GA, respectively. For the latter, we also list the disks covered and the percentage gap with respect to the nearOpt solution.

GA correctly identifies all solutions of the convex hull instances. Nevertheless, we cannot properly evaluate the performance of our genetic algorithm from these instances, as they are easy to solve since the number of customers to cover varies from only 8 to 13. Therefore, we have performed further tests on more complex instances, intending to evaluate our approach properly. For this purpose, we used the adapted instances of CETSP. For the definition of high-quality solutions, we used nearOpt. Given the higher complexity of the problem, we gave the procedure several sequences for each instance to explore

Instances	**nearOpt**		**GA**					
	value	*time*	*value*	*time*	*mid*	*inner*	*outer*	*GAP*
hull1	1275.40	0.03	1275.40	0.60	0	1	10	0.00%
hull2	1485.44	0.03	1485.44	0.66	0	2	8	0.00%
hull3	2955.62	0.03	2955.62	1.13	0	4	5	0.00%
hull4	1536.79	0.03	1536.79	0.86	0	2	9	0.00%
hull5	1369.55	0.03	1369.55	0.55	0	2	9	0.00%
hull6	2893.97	0.03	2893.97	1.44	0	6	7	0.00%
hull7	1400.98	0.03	1400.98	0.58	0	2	11	0.00%
hull8	1484.06	0.03	1484.06	0.46	0	1	7	0.00%
hull9	1500.26	0.03	1500.26	0.55	0	6	7	0.00%

Table 4.5: Comparison between nearOpt and GA on convex hull instances

the solution space in depth. A sequence can be obtained by solving a TSP on the instance. To obtain different ones, we perturbed the instances. Specifically, let E be the set of arcs defined by the instance, and $w(e)$ the weight of the arc $e \in E$, $w(e) = w(e) \times f$, where f is a random number in the interval $[1 - eps, 1 + eps]$. The parameter *eps* is the degree of perturbation applied to the instance. Trivially, the greater its value, the greater the difference between the produced sequences. Nevertheless, if we excessively increase the value *eps*, the solutions produced would be of poor quality since the instance is spoiled compared to the original.

In our experiments, we computed 100 different sequences with $eps = 0.1$ and solved the TSPs using the LKH algorithm proposed by [Lin and Kernighan, 1973]. Then, we obtained a set of GCETSP solutions from the computed sequences by applying nearOpt on them. From this set, we preserve the best one. Finally, we compared the GA solutions with the nearOpt solutions. The results are shown in the tables 4.6, 4.7 and 4.8. The tables are organized as follows: the first column reports the instance name, while in the next two, we report information about the solutions computed by GA and nearOpt, respectively. For both approaches, we show the solution's value; the computational times in seconds; the disks selected from a solution, divided as mid, inner, outer; and the percentage gap from the best solution found. The gap is obtained by applying the formula: $\frac{UB-Best}{Best}$, where UB is the solution value of the approach, and $Best$ is the best solution found for that instance.

Instances	**GA**						**nearOpt**					
	value	*time*	*mid*	*inner*	*outer*	*gap*	*value*	*time*	*mid*	*inner*	*outer*	*gap*
bonus1000	1932.93	13886.54	142	824	34	7.07%	**2080.05**	59.88	1	984	15	0.00%
bubbles1	**-248.90**	1.32	4	19	13	0.00%	-269.76	2.42	3	20	13	8.38%
bubbles2	**-241.70**	6.76	24	31	21	0.00%	-280.09	5.05	10	57	9	15.89%
bubbles3	**-269.16**	22.80	56	40	30	0.00%	-319.06	7.33	17	21	88	18.54%
bubbles4	**-308.29**	56.41	60	54	70	0.00%	-502.06	10.49	23	44	117	62.86%
bubbles5	**-458.73**	88.71	65	88	97	0.00%	-551.72	13.80	15	33	202	20.27%
bubbles6	**-560.34**	257.61	89	109	126	0.00%	-728.67	17.74	21	52	251	30.04%
bubbles7	**-621.24**	443.75	123	130	153	0.00%	-884.76	20.24	23	66	317	42.42%
bubbles8	**-817.96**	776.32	146	169	181	0.00%	-1143.11	24.55	33	65	398	39.75%
bubbles9	**-918.40**	1145.50	153	208	233	0.00%	-1347.74	30.41	62	132	400	46.75%
chaoSingleDep	**-547.47**	24.28	25	175	0	0.00%	-581.36	10.64	1	138	61	6.19%
concentricCircles1	**-18.48**	0.47	1	12	3	0.00%	-20.42	1.99	1	13	2	10.51%
concentricCircles2	**-67.31**	1.33	1	23	12	0.00%	-78.30	2.72	1	30	5	16.32%
concentricCircles3	**-131.17**	4.47	1	35	24	0.00%	-153.29	3.50	1	42	17	16.86%
concentricCircles4	**-210.62**	14.61	9	61	34	0.00%	-254.89	6.31	1	84	19	21.02%
concentricCircles5	**-298.69**	18.03	8	86	54	0.00%	-347.56	8.21	1	100	47	16.36%
rotatingDiamonds1	**17.21**	0.67	0	20	0	0.00%	**17.21**	1.72	1	19	0	0.00%
rotatingDiamonds2	**18.84**	1.00	0	60	0	0.00%	**18.84**	3.27	1	59	0	0.00%
rotatingDiamonds3	**127.36**	6.57	0	180	0	0.00%	127.33	9.91	1	179	0	0.02%
rotatingDiamonds4	**146.24**	24.41	0	320	0	0.00%	**146.23**	17.17	1	319	0	0.01%
rotatingDiamonds5	**474.79**	159.44	0	680	0	0.00%	474.59	36.14	1	679	0	0.04%
team1_100	**-69.02**	9.63	19	72	9	0.00%	-130.52	7.80	10	20	70	89.09%
team2_200	**143.62**	44.87	71	116	10	0.00%	60.37	13.08	57	143	0	57.96%
team3_300	**307.65**	136.02	22	243	34	0.00%	100.16	18.56	11	166	123	67.44%
team4_400	**268.57**	441.42	31	325	44	0.00%	155.05	22.52	14	313	73	42.27%
team5_499	**614.30**	384.72	2	468	29	0.00%	586.33	24.61	1	490	8	4.55%
team6_500	**979.56**	2106.47	58	442	0	0.00%	956.76	26.24	4	496	0	2.33%

Table 4.6: Comparison between GA and nearOpt on instances with varied overlap ratios

Instances	**GA**						**nearOpt**					
	value	*time*	*mid*	*inner*	*outer*	*gap*	*value*	*time*	*mid*	*inner*	*outer*	*gap*
Overlap ratio 0.02												
d493	**1204.88**	320.38	0	492	0	0.00%	1175.24	27.53	1	491	0	2.54%
dsj1000	1560.87	1726.90	21	900	78	3.81%	**1622.71**	60.23	1	980	18	0.00%
kroD100	**105.23**	3.68	0	99	0	0.00%	103.54	6.62	1	98	0	1.60%
lin318	**-1809.38**	169.88	40	146	131	0.00%	-2010.03	16.27	28	87	202	11.09%
pcb442	**901.54**	178.69	0	441	0	0.00%	878.10	22.48	1	440	0	2.60%
rat195	**376.20**	26.20	0	194	0	0.00%	371.01	10.62	1	193	0	1.38%
rd400	**-35.02**	456.07	1	320	78	0.00%	-102.18	19.55	1	377	21	191.76%
Overlap ratio 0.1												
d493	**1306.76**	614.51	1	491	0	0.00%	1291.89	26.96	1	491	0	1.14%
dsj1000	1833.93	2135.72	198	733	68	9.45%	**2025.38**	61.82	99	858	42	0.00%
kroD100	**162.64**	10.99	2	96	1	0.00%	159.61	6.47	1	98	0	1.87%
lin318	**-559.55**	209.08	98	148	71	0.00%	-751.58	16.77	67	40	210	34.32%
pcb442	**1061.84**	566.24	4	436	1	0.00%	1025.21	22.60	1	440	0	3.45%
rat195	**456.06**	65.64	0	194	0	0.00%	436.20	11.36	1	193	0	4.36%
rd400	**379.11**	717.80	84	298	17	0.00%	287.95	21.47	22	311	66	24.04%
Overlap ratio 0.3												
d493	1365.03	934.31	2	490	0	0.11%	**1366.56**	28.86	1	491	0	0.00%
dsj1000	2222.68	2353.69	159	801	39	12.86%	**2550.70**	62.60	1	998	0	0.00%
kroD100	**207.29**	6.51	0	99	0	0.00%	205.17	6.19	1	98	0	1.02%
lin318	**-195.36**	127.35	108	194	15	0.00%	-364.30	17.42	114	148	55	86.48%
pcb442	**1173.68**	510.40	1	440	0	0.00%	1115.19	22.21	1	440	0	4.98%
rat195	**513.81**	47.05	0	194	0	0.00%	502.97	10.78	1	193	0	2.11%
rd400	**699.41**	757.13	60	336	3	0.00%	678.30	21.92	1	398	0	3.02%

Table 4.7: Comparison between GA and nearOpt on instances with fixed overlap ratios

Instances	**GA**						**nearOpt**					
	value	*time*	*mid*	*inner*	*outer*	*gap*	*value*	*time*	*mid*	*inner*	*outer*	*gap*
bonus1000rdmRad	**1647.95**	4011.67	50	902	48	0.00%	1607.94	58.01	11	981	8	2.43%
d493rdmRad	**1295.73**	27.56	0	490	2	0.00%	1267.29	26.33	1	491	0	2.20%
dsj1000rdmRad	**2176.49**	186.28	36	943	20	0.00%	2000.15	60.60	2	984	13	8.10%
kroD100rdmRad	**119.58**	8.73	0	98	1	0.00%	118.60	6.88	1	97	1	0.82%
lin318rdmRad	**-1087.01**	55.98	34	224	59	0.00%	-1485.95	16.17	19	87	211	36.70%
pcb442rdmRad	**1024.94**	108.44	1	440	0	0.00%	983.34	21.27	1	440	0	4.06%
rat195rdmRad	**498.64**	31.27	1	193	0	0.00%	480.54	10.50	1	193	0	3.63%
rd400rdmRad	**-208.05**	235.30	0	367	32	0.00%	-228.03	19.28	1	396	2	9.60%
team1_100rdmRad	**-125.70**	11.21	12	79	9	0.00%	-164.53	6.78	2	61	37	30.89%
team2_200rdmRad	**-131.55**	82.33	19	146	35	0.00%	-208.70	11.35	8	156	36	58.65%
team3_300rdmRad	**427.20**	211.46	30	259	11	0.00%	318.25	15.96	12	203	85	25.50%
team4_400rdmRad	**18.28**	228.07	5	355	40	0.00%	-18.06	20.28	1	389	10	198.81%
team5_499rdmRad	**979.18**	22.85	14	470	15	0.00%	894.31	25.44	6	486	7	8.67%
team6_500rdmRad	**651.57**	144.84	39	427	34	0.00%	541.73	24.29	17	479	4	16.86%

Table 4.8: Comparison between GA and nearOpt on instances with random radii

The genetic algorithm identifies better solutions in most of the benchmarks considered, with gaps ranging from a low of 0.82% (*kroD100rdmRad*) to a high of 198.81% (*team4_400rdmRad*). On large instances, GA produces inferior solutions, while nearOpt succeeds better, such as on *bonus1000* and *dsj1000* with different overlap ratios. Furthermore, the solutions produced by nearOpt are comparable to those of GA, except for a few instances where the difference is substantial. Let's examine the morphology of the instances, such as *bubbles* and *teams* that we show in Figure 4.7. We can see that they have a high degree of overlap between targets, leading the nearOpt approach to producing poor solutions. It is reasonable since the algorithm is constrained by how the points on each disk are identified. As described above, we use SOCP by considering one disk at a time for point identification. In the case of highly overlapped instances, SOCP generally produces points that match each other, especially on external disks. This scenario leads the nearOpt to prefer external disks without considering the possibility of inward deviation since a small number of turning points visit several customers. As we can observe from the table, for the instances *bubbles* and *teams* nearOpt prefers external disks. At the same time, GA is more open to deviating inward, resulting in a larger number of intermediate disks traversed than nearOpt.

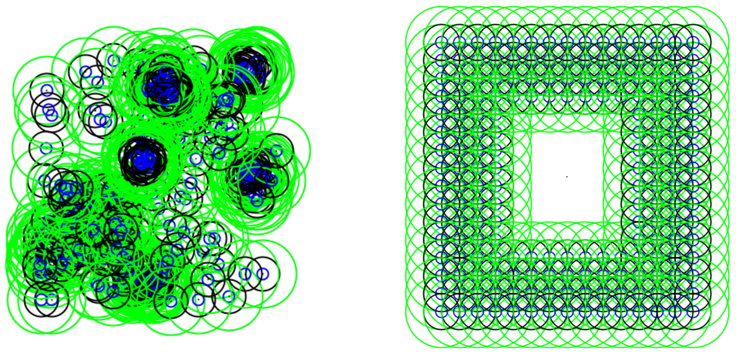


Figure 4.7: Instance *team3_300* (left) and *bubbles4* (right)

nearOpt into GA: From the above results, we can see that, in general, nearOpt produces relatively high-quality solutions with significantly reduced time compared to the genetic algorithm. Based on this, we conducted additional tests to see if it could incorporate nearOpt into the GA pipeline. The goal is to improve the quality of solutions produced by the GA by further reducing computational time. We performed two tests: the first one sees the inclusion of the solutions produced by nearOpt during the generation of the initial population. In contrast, the second one uses the approach as an additional improvement procedure. In the first case, it was necessary to determine a set of solutions to be included that had sufficient variance between them to preserve variety in the population. Thus, we need to identify which and how many

	50		**40**		**25**		**10**		**0**	
	gap	*Time*	*gap*	*Time*	*gap*	*Time*	*gap*	*Time*	*gap*	*Time*
Len sort	0.27%	655.62	0.79%	502.72	0.36%	620.17	0.33%	766.11	1.48%	602.71
FF sort	0.52%	516.49	0.13%	467.02	0.43%	439.55	0.39%	561.63	0.51%	602.71
Prize sort	0.53%	525.7	1.57%	429.15	0.65%	474.27	0.94%	494.02	0.78%	602.71
Avg	0.44%	565.94	0.83%	466.30	0.48%	511.33	0.55%	607.25	0.92%	602.71

Table 4.9: Comparison of GA solutions with different combinations of random and nearOpt solutions in the initial population generation. The rows specify different sorting criteria for the nearOpt solution selection: total length (Len sort), fitness function value (FF sort), and prize collected (Prize sort). The columns show the number of nearOpt solutions included in the GA initial population, from 50 to 0.

solutions to include. As for how many solutions to add, we can determine the number empirically by testing different combinations of random and nearOpt solutions: given $pop_{size} = 50$, the combinations of random-nearOpt solutions considered are 0-50, 10-40, 25-25, 40-10, and 50-0. Regarding which solutions to add, we can identify three different criteria for evaluating solutions: objective function value, total length, and collected reward. Trivially, these criteria reflect the different possibilities of distinction. Sorting by collected reward means favoring a diversification in terms of disks crossed since the reward depends on the choice of disks. On the contrary, sorting by total route length means favoring a diversification in terms of sequences since they affect the route more. Finally, sorting by objective function value combines the previous two.

We tested the different combinations described above for each criterion. To maximize the variance among the included nearOpt solutions, we select solutions sequentially by alternating heads and tails from a set of 100 solutions generated by the approach. The results are reported in the table 4.9, organized as follows. The first column reports the different sorting criteria considered for the nearOpt solutions, which are *Len sort*, *FF sort*, and *Prize sort*. In the *Len sort* row, we report the nearOpt solutions sorted by route length, while the *FF sort* row relates to the solutions sorted by fitness value. Finally, in *Prize sort* we show the solutions of nearOpt sorted by total prize collected. The remaining columns illustrate the averages of GA gaps and times in seconds with different numbers of nearOpt solutions included: 50, 40, 25, 10, and 0, respectively. We can infer that no combination prevails over the others from the results. We do not find significant improvements over the original GA regarding the computational times.

We performed several tests to ascertain the feasibility of including nearOpt in the genetic algorithm as an additional improvement procedure. Specifically, we included nearOpt in the GA instead of improveSolution. The results show that the original version performs significantly better than the modified version of the solution produced. While, in terms of computational time, the modified version performs better on several instances, probably because it converges to a local optimum first and thus is prevented from further exploring the search space.

4.5 Conclusions

In recent years, the close enough traveling salesman problem has been widely discussed for its ability to model several real-world applications, especially concerning drones, such as meter reading. The goal of CETSP is to identify the shortest crossing of all neighborhoods associated with a given target set, starting and ending at the depot. In this part of the dissertation, we presented a new generalization called generalized close enough traveling salesman problem. Each customer is associated with a set of concentric disks with different radii. Each disk is associated with a premium that decreases as the distance from the center increases. Introducing different disks around the customer allows us to model real-world applications to get better results by getting closer to the target. For example, in RFID system applications, the rewards can represent the probability of success in reading the meter of a generic target. The GCETSP solution determines the admissible route that maximizes the difference between the total collected reward and the route length. The admissible route means exactly visiting a customer disk and the depot.

We propose benchmark instances for the problem, either defined based on those present for CETSP by adding disks for each target or generated from scratch. In addition, we propose an adapted version of the genetic algorithm presented for CETSP, which provides high-quality solutions for GCETSP.

We performed tests to highlight the differences between the CETSP and GCETSP solutions. We compared three different CETSP solutions, one for each disk, with the one produced by the adapted GA. The results reveal that a good solution for CETSP is far from suitable for its generalization. Although it can be seen that CETSP solutions considering a single disk are relatively good, it is necessary to consider a combination of disks to obtain a high-quality solution.

Finally, we conducted a series of tests to verify the quality of the solutions produced by the GA. We compared the genetic algorithm with a heuristic approach called nearOpt, which identifies the best turning points for each target from a given visit sequence. We compared the two approaches on instances where nearOpt should provide optimal (or near-optimal) solutions and observed that the GA always defines the same solution. Next, we compared them on more challenging instances and found that the GA can identify reasonable solutions on these instances, outperforming nearOpt, despite the approaches being comparable in most cases.

Chapter 5

Conclusions and future works

In this dissertation, we have discussed in depth the close enough traveling salesman problem(CETSP), defining advanced solving methods and describing new variants of the problem, such as the mixed constrained generalized routing problem (MGCRP) and the generalized close enough traveling salesman problem(GCETSP). CETSP is a variant of the classical TSP, in which one must find the Hamiltonian cycle that passes at a given distance for each target. We presented a solving metaheuristic for this problem, specifically a genetic algorithm. The GA uses the concept of a turning point to define the solution, i.e., a moving point associated with a target, with the characteristic of covering that target. At its core, it combines two local searches, a 2-opt to improve the crossing sequence and 3Alg to optimize the position of the points of a fixed sequence, and a mathematical model, the SOCP, to provide optimal solutions. Our approach provides the best solutions for 59 out of 62 cases on the reference instances proposed by [Mennell, 2009]. In addition, we formulated two novel metrics, the TSPD and the OC, which describe the difficulty of the problem viewing the morphology of the instances and the conformation of the solution, respectively. TSPD is based on the ratio between the targets' circumference radii and their distances to estimate the degree of overlap between them. OC estimates the difficulty of the instance for the number of points of which the solution is composed. We compared our metrics with the one already present in the literature, the overlap ratio, obtaining that our proposed metrics provide more information about the complexity of the problem. Furthermore, we found errors in the OR values for some instances found in the literature and presented the corrected ones in comparing the metrics. Finally, we presented a real application case of CETSP in the context of a research project at the University of Molise, where it was necessary to define the route of a drone to take thermal pictures of photovoltaic panels. We modeled the panels as targets, for which there is a zone within which it is possible to photograph. The zone is defined following the panel's inclination and the flight altitude of the drone to be shifted concerning the original panel's position. Before using our approach, the route was computed as the solution of a TSP. With GA, we found a route savings of 15%, with no loss of quality in the images produced. Also, as part of the research project, we produced a framework for producing image convolution filters. It was accomplished by formulating the problem as a filter retrieval problem and modeling it mathematically. We proposed three models, one of linear programming (LP) and two of integer linear programming (ILP). We tested them on different application scenarios, such as the recognition of convolution filters with clear or noises-affected images and the recognition of filters defined ad hoc for feature enhancement. The results indicate that our modeling, especially the ILP one, can correctly identify all the proposed convolution filters in the absence of perturbations. In contrast, in the case of noise-affected images, the identified filters produce images with a $MAE \leq 4$ compared to the original output image. In the case of feature highlighters filters, the models produce reasonably good filters. Finally, having formulated the problem mathematically, our approaches certify the presence of a convolution filter $k \times h$ capable of emphasizing the desired feature.

We have seen how CETSP models several real-world application scenarios, especially related to drones. However, these scenarios are not always accurate, as the drone is not always free to fly. We presented a generalization of CETSP and CEARP, named as mixed constrained generalized routing problem. We introduce the concept of a flight zone with different degrees of flight freedom. We specifically distinguish two: the free-flight zone (FFZ), which the drone is free to fly, and the constrained flight zone (CFZ), where there are limitations or prohibitions. The above problem has only been examined in one of its limiting cases, i.e., when only FFZs are present. Future developments include studying the second limiting case as well in order to combine the solving techniques to define a mixed one.

Furthermore, we presented a second new generalization of CETSP, called the generalized close enough traveling salesman problem (GCETSP). In the case of CETSP, every customer is associated with a single area that, if traversed, is considered covered. In GCETSP, each customer is associated with several concentric areas of different radii. A premium is associated with each area that grows as the radius decreases, collected when the area is crossed. Introducing different neighborhoods for each customer allows modeling of different real-world application contexts, in which we get greater rewards if we approach close to the targets. For example, when considering RFID meter reading systems, the rewards can represent the probability of successful meter reading. This probability decreases as we move away from the customer, which explains the decreasing reward with respect to the length of the disk radius. For this problem, we propose benchmark instances based on those present for CETSP with the addition of disks for each customer and generated from scratch. In addition, we propose an adapted version of the genetic algorithm presented for CETSP and a constructive heuristic. We performed tests to examine the differences between the classical problem, CETSP, and this new generalization, GCETSP. We compared several CETSP solutions, one on each disk, with the one produced by the adapted GA. The results reveal that a good solution for CETSP is not the same for its generalization, but a mixed disk combination is needed. We conducted a series of tests to verify the quality of the solutions produced by our approach. We compared the GA with the constructive heuristic called nearOpt, which identifies the best turning points for each customer given a predefined visit sequence. We compared the two approaches on instances where nearOpt provides optimal or near-optimal solutions, and the GA always identifies the same solution. Finally, we compared the two approaches on more complex instances and found that GA identifies better solutions in most cases, outperforming nearOpt.

This dissertation is structured to promote further study of CETSP and its variants, given the emerging use of drones in logistics and transportation. Future developments are planned in this direction: as, for example, GCETSP, we are studying variants of the previously proposed generalization, in which constraints are imposed on the route definition. For example, referring purely to drones, the route definition can vary depending on their battery. The drone can cover more customers if the time at its disposal is more significant, i.e., if the battery allows it to do so. Moreover, the definition of the route can vary according to the amount of prize we desire to collect. We can condition the choice of disks if we need to collect a specific prize. Based on this, we define two variants of GCETSP: one aims to determine an eligible route that maximizes the total prize collected while restraining the route length to a certain threshold. By an eligible route for this variant, we refer to a route that begins and ends in the depot and does not exceed a given threshold. Note that it is possible to prevent some customers from being considered in this case. The second one aims to determine a feasible route that minimizes the route length while collecting a specific prize. By eligible route for this variant, we mean a route that visits precisely one record per customer and depot. We have developed preliminary approaches that produce reasonable admissible solutions to these problems.

Bibliography

[Alpaydin, 2009] Alpaydin, E. (2009). *Introduction to machine learning.* MIT press.

[Arkin and Hassin, 1994] Arkin, E. M. and Hassin, R. (1994). Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55(3):197–218.

[Behdani and Smith, 2014] Behdani, B. and Smith, J. C. (2014). An integer-programming-based approach to the close-enough traveling salesman problem. *INFORMS Journal on Computing*, 26(3):415–432.

[Booker, 1987] Booker, L. (1987). Improving search in genetic algorithms. *Genetic algorithms and simulated annealing*, pages 61–73.

[Bovik, 2010] Bovik, A. C. (2010). *Handbook of image and video processing.* Academic press.

[Capobianco et al., 2021] Capobianco, G., Cerrone, C., Di Placido, A., Durand, D., Pavone, L., Russo, D. D., and Sebastiano, F. (2021). Image convolution: a linear programming approach for filters design. *Soft Computing*, pages 1–16.

[Carrabs et al., 2017a] Carrabs, F., Cerrone, C., Cerulli, R., and D'Ambrosio, C. (2017a). Improved upper and lower bounds for the close enough traveling salesman problem. In *Proceedings of International Conference on Green, Pervasive, and Cloud Computing*, pages 165–177. Springer.

[Carrabs et al., 2017b] Carrabs, F., Cerrone, C., Cerulli, R., and Gaudioso, M. (2017b). A novel discretization scheme for the close enough traveling salesman problem. *Computers & Operations Research*, 78:163–171.

[Carrabs et al., 2020] Carrabs, F., Cerrone, C., Cerulli, R., and Golden, B. (2020). An adaptive heuristic approach to compute upper and lower bounds for the close-enough traveling salesman problem. *INFORMS Journal on Computing*, 32(4):1030–1048.

[Cerrone et al., 2017a] Cerrone, C., Cerulli, R., and Golden, B. (2017a). Carousel greedy: a generalized greedy algorithm with applications in optimization. *Computers & Operations Research*, 85:97–112.

[Cerrone et al., 2017b] Cerrone, C., Cerulli, R., Golden, B., and Pentangelo, R. (2017b). A flow formulation for the close-enough arc routing problem. In *International Conference on Optimization and Decision Science*, pages 539–546. Springer.

[Chen and Fomel, 2015] Chen, Y. and Fomel, S. (2015). Random noise attenuation using local signal-and-noise orthogonalization. *Geophysics*, 80(6):WD1–WD9.

[Coutinho et al., 2016] Coutinho, W. P., Nascimento, R. Q. d., Pessoa, A. A., and Subramanian, A. (2016). A branch-and-bound algorithm for the close-enough traveling salesman problem. *INFORMS Journal on Computing*, 28(4):752–765.

[Current and Schilling, 1989] Current, J. R. and Schilling, D. A. (1989). The covering salesman problem. *Transportation science*, 23(3):208–213.

[Dhawan et al., 1985] Dhawan, A. P., Rangayyan, R. M., and Gordon, R. (1985). Image restoration by wiener deconvolution in limited-view computed tomography. *Applied optics*, 24(23):4013–4020.

[Di Placido et al., 2021] Di Placido, A., Archetti, C., and Cerrone, C. (2021). A genetic algorithm for the close-enough traveling salesman problem with application to solar panels diagnostic reconnaissance. *accepted to Computers & Operations Research.*

[Dong et al., 2007] Dong, J., Yang, N., and Chen, M. (2007). Heuristic approaches for a tsp variant: The automatic meter reading shortest tour problem. In *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, pages 145–163. Springer.

[Drexl, 2014] Drexl, M. (2014). On the generalized directed rural postman problem. *Journal of the Operational Research Society*, 65(8):1143–1154.

[Dumitrescu and Mitchell, 2003] Dumitrescu, A. and Mitchell, J. S. (2003). Approximation algorithms for tsp with neighborhoods in the plane. *Journal of Algorithms*, 48(1):135–159.

[Gendreau et al., 1997] Gendreau, M., Laporte, G., and Semet, F. (1997). The covering tour problem. *Operations Research*, 45(4):568–576.

[Gulczynski et al., 2006] Gulczynski, D. J., Heath, J. W., and Price, C. C. (2006). The close enough traveling salesman problem: A discussion of several heuristics. In Alt, F. B., Fu, M. C., and Golden, B. L., editors, *Perspectives in Operations Research*, pages 271–283. Springer.

[Ha et al., 2012] Ha, M. H., Bostel, N., Langevin, A., and Rousseau, L.-M. (2012). An exact algorithm for the close enough traveling salesman problem with arc covering constraints. In *ICORES*, pages 233–238.

[Ha et al., 2014] Ha, M. H., Bostel, N., Langevin, A., and Rousseau, L.-M. (2014). Solving the close-enough arc routing problem. *Networks*, 63(1):107–118.

[Holland et al., 1992] Holland, J. H. et al. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

[Kennedy and Eberhart, 1995] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE.

[Lin and Kernighan, 1973] Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516.

[Mata and Mitchell, 1995] Mata, C. S. and Mitchell, J. S. (1995). Approximation algorithms for geometric tour and network design problems. In *Proceedings of the eleventh annual symposium on Computational geometry*, pages 360–369.

[Mennell et al., 2011] Mennell, W., Golden, B., and Wasil, E. (2011). A Steiner-zone heuristic for solving the close-enough traveling salesman problem. In *Proceedings of 2th INFORMS computing society conference: operations research, computing, and homeland defense.*

[Mennell, 2009] Mennell, W. K. (2009). *Heuristics for solving three routing problems: Close-enough traveling salesman problem, close-enough vehicle routing problem, sequence-dependent team orienteering problem.* PhD thesis, University of Maryland (College Park, Md.).

[Michailovich and Tannenbaum, 2007] Michailovich, O. and Tannenbaum, A. (2007). Blind deconvolution of medical ultrasound images: A parametric inverse filtering approach. *IEEE Transactions on Image Processing*, 16(12):3005–3019.

[Mitchell, 1998] Mitchell, M. (1998). *An introduction to genetic algorithms.* MIT press.

[Moscato et al., 1989] Moscato, P. et al. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989.

[Nussbaumer, 2012] Nussbaumer, H. J. (2012). *Fast Fourier transform and convolution algorithms*, volume 2. Springer Science & Business Media.

[Poikonen et al., 2017] Poikonen, S., Wang, X., and Golden, B. (2017). The vehicle routing problem with drones: Extended models and connections. *Networks*, 70(1):34–43.

[Sadek, 2012] Sadek, R. A. (2012). Svd based image processing applications: state of the art, contributions and research challenges. *arXiv preprint arXiv:1211.7102*.

[Shuttleworth et al., 2008] Shuttleworth, R., Golden, B. L., Smith, S., and Wasil, E. (2008). Advances in meter reading: Heuristic solution of the close enough traveling salesman problem over a street network. In Golden, B., Raghavan, S., and Wasil, E., editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 487–501. Springer.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Wang et al., 2019] Wang, X., Golden, B., and Wasil, E. (2019). A Steiner zone variable neighborhood search heuristic for the close-enough traveling salesman problem. *Computers & Operations Research*, 101:200–219.

[Wu et al., 2017] Wu, Y., Zhang, Y., Zhang, Y., Huang, Y., and Yang, J. (2017). Tsvd with least squares optimization for scanning radar angular super-resolution. In *2017 IEEE Radar Conference (RadarConf)*, pages 1450–1454. IEEE.

[Xu et al., 1994] Xu, C., Aissaoui, I., and Jacquey, S. (1994). Algebraic analysis of the van cittert iterative method of deconvolution with a general relaxation factor. *JOSA A*, 11(11):2804–2808.

[Yang et al., 2018] Yang, Z., Xiao, M.-Q., Ge, Y.-W., Feng, D.-L., Zhang, L., Song, H.-F., and Tang, X.-L. (2018). A double-loop hybrid algorithm for the traveling salesman problem with arbitrary neighbourhoods. *European Journal of Operational Research*, 265(1):65–80.

[Yuan et al., 2007] Yuan, B., Orlowska, M., and Sadiq, S. (2007). On the optimal robot routing problem in wireless sensor networks. *IEEE transactions on knowledge and data engineering*, 19(9):1252–1261.

[Zhao and Desilva, 1998] Zhao, F. and Desilva, C. J. (1998). Use of the laplacian of gaussian operator in prostate ultrasound image processing. In *Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. Vol. 20 Biomedical Engineering Towards the Year 2000 and Beyond (Cat. No. 98CH36286)*, volume 2, pages 812–815. IEEE.