Applied Soft Computing Journal 153 (2024) 111266

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

An effective memetic algorithm for the close-enough traveling salesman problem

Zhenyu Lei, Jin-Kao Hao *

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France

ARTICLE INFO

Keywords:
Traveling salesman problem
Close-enough traveling salesman problem
Combinatorial optimization
Heuristics

ABSTRACT

The Close-Enough Traveling Salesman Problem (CETSP) is a variant of the well-known Traveling Salesman Problem (TSP). Unlike the TSP, each target in the CETSP has a disk neighborhood, and the target is considered visited when any point in its neighborhood is visited. This feature makes the CETSP a suitable model for many real-world applications. In this work, we propose an effective memetic algorithm that integrates a carefully designed crossover operator and an effective local optimization procedure with original search operators. Experimental results on the 62 well-known benchmark instances show that the algorithm is highly competitive with the state-of-the-art methods, reporting 30 new best upper bounds. We demonstrate the usefulness of the algorithm on a real laser welding robot path planning problem. We provide insights into the understanding of the algorithm design.

1. Introduction

The Close-Enough Traveling Salesman Problem (CETSP) [1] is a variant of the popular Traveling Salesman Problem (TSP). The CETSP can be defined as follows. Given a set of N targets $V = \{v_1, v_2, \ldots, v_N\}$ and a depot p_0 in the Euclidean plane, each target v_i has a disk neighborhood $\mathcal{N}_i$ of radius r_i. The objective is to find the shortest Hamiltonian cycle $S = \{p_0, p_1, p_2, \ldots, p_N, p_0\}$ that starts and ends at the depot p_0 and passes through a point (also called position hereafter) p_i in the disk neighborhood $\mathcal{N}_i$ of every target v_i. Let $d(x, y)$ be the distance between points x and y. The CETSP is defined as follows.

$$\text{(CETSP)} \quad \text{Min} \quad f(S) = \sum_{i=0}^{N-1} d(p_i, p_{i+1}) + d(p_N, p_0), \tag{1}$$

$$\text{subject to} \quad S = \{p_0, p_1, \ldots, p_N, p_0\}, p_i \in \mathcal{N}_i, i = 1, \ldots, N$$

The CETSP is a complex problem that combines the discrete optimization of the visiting sequence, as in the TSP, with the continuous optimization of the visiting positions for the targets. In contrast to the standard TSP, where the salesman must visit the targets, in the CETSP, it is only necessary to pass through any point within the disk neighborhood of each target. The special feature of the CETSP may result in shorter tours than the standard TSP, which is useful in many practical situations. For instance, in the graph of Fig. 1, the black triangle indicates the depot, the black nodes are the targets with their disk neighborhood. For this example, the CETSP tour is obviously shorter than the TSP tour. The CETSP can be seen as a generalization of the TSP. In fact, if all the disk neighborhoods have a radius of 0, the CETSP reduces to the standard TSP. As a result, the CETSP is an NP-hard problem and is computationally difficult to solve [2].

The CETSP occurs naturally in various real-world applications, such as automated meter reading with radio frequency identification (RFID) [1], solar panel diagnostic reconnaissance [3], and laser welding robot path planning [4]. With the development of unmanned aerial vehicles (UAVs), the CETSP can be extended to model various missions, such as aerial reconnaissance, supply delivery, and pipeline surveillance, etc.

Given the relevance of the CETSP, a number of solution methods have been developed. In Section 2, we provide a literature review of the related works. Since the problem was introduced in [1], there has been a steady stream of important progress. However, it has been observed that even the best-performing methods do not perform consistently well on the existing benchmark instances and may require considerable computational time to achieve their best results. In this work, we aim to advance the state-of-the-art in effectively solving this challenging problem and to provide a useful tool that can benefit researchers and practitioners working on the CETSP and related problems. To this end, we introduce a highly effective memetic algorithm for the CETSP that combines population-based genetic search with local optimization based on variable neighborhood descent.

Our contributions are twofold.

- The algorithm features a dedicated crossover operator that is able to inherit good features from parent solutions and guide

* Corresponding author.
E-mail addresses: zhenyu.lei@etud.univ-angers.fr (Z. Lei), jin-kao.hao@univ-angers.fr (J.-K. Hao).

https://doi.org/10.1016/j.asoc.2024.111266
Received 29 July 2023; Received in revised form 20 December 2023; Accepted 7 January 2024
Available online 26 January 2024

Table 1
Summary of the algorithms for CETSP in the literature.

Algorithm	Year	Method	Characteristics
Gulczynski et al. [1]	2006	Heuristics	3-phase heuristics, Steiner-zone
Mennell et al. [2,5]	2009, 2011	Heuristics	3-phase heuristics, Steiner-zone, SOCP
Behdani and Smith [6]	2014	Exact algorithm	MIP, discretization
B&B [7]	2016	Exact algorithm	branch and bound, SOCP
Carrabs et al. [8]	2017	Heuristics	discretization, MIP
Carrabs et al. [9]	2017	Heuristics	discretization, SOCP
HA [10]	2018	Heuristics	PSO, GA
SZVNS [11]	2019	Heuristics	VNS, Steiner-zone
(ul/bl)Alg [12]	2020	Heuristics	discretization, carousel greedy algorithm
GA [3]	2022	Heuristics	GA, SOCP

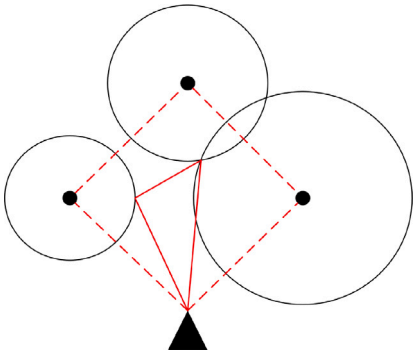


Fig. 1. Example of a CETSP tour (solid line) and a standard TSP tour (dotted line).

the algorithm to promising search regions. Its local optimization component is based on the VND method and integrates new effective local search operators, i.e., the disk-constrained position optimization and the joint optimization of sequence and position.

- We report 30 new best solutions (updated upper bounds) and 32 equal best-known solutions including 23 known optimal solutions, for a total of 62 benchmark instances tested in the literature. In addition, we showcase the usefulness of the algorithm on a real laser welding robot path planning problem. We perform additional experiments and analyses to shed light on the search components of the proposed algorithm. We make the code of the algorithm available online to enable its use in research and applications.

The remainder of this paper is structured as follows. Section 2 provides a literature review on the CETSP. Section 3 presents the proposed algorithm for solving the CETSP. Section 4 shows computational results on benchmark instances and comparisons with state-of-the-art methods. Section 5 examines the application of the proposed algorithm to a real-world problem. Section 6 provides an analysis of the main components of the algorithm. Section 7 concludes the paper and discusses future research directions.

2. Literature review

The CETSP has attracted significant research interest over the past two decades.

Table 1 summarizes the solution methods for the CETSP in the literature. With the introduction of the CETSP, Gulczynski et al. [1] (2016) presented six heuristics (Steiner-zone, sweeping circle, radial adjacency, etc.) to solve the problem. These heuristics follow three common steps: (1) find a feasible supernode set S, (2) generate a feasible tour T on the points in S, and (3) improve the found feasible tour T. These algorithms formed the first foundation for solving the CETSP heuristically.

Building on this work, a three-phase Steiner-zone heuristic [2,5] (2009, 2011) was proposed to solve the CETSP. This heuristic first identifies a set of Steiner zones covering all targets, which are the non-empty intersections of disk neighborhoods. It then discretizes these Steiner zones to obtain a set of points S and generates a feasible tour T by solving the TSP in S. Finally, the found feasible tour T is improved by modifying the positions of the discretization points. In the third phase, the problem is formulated as a Touring Steiner Zones Problem and solved by a powerful optimization technique called the second-order cone programming (SOCP) [13]. By decomposing the problem into sub-problems, this three-phase approach helps to manage the complexity of the overall problem. However, the solution quality of this approach depends heavily on the quality of the supernodes, and the search can be trapped in poor local optima.

In 2019, Wang et al. [11] also used the concept of Steiner-zones and proposed a fast three-step heuristic (SZVNS) based on the Variable Neighborhood Search (VNS) approach to solve the CETSP. The first step involves data cleaning to reduce the problem size and to improve the running time. The second step constructs Steiner-zones and generates a feasible tour that passes through all Steiner-zones. The third step uses iterative variable neighborhood search to improve the feasible solution. This is achieved by applying six operators, including TSP heuristics, Steiner point selection, one-point insertion, two-point insertion, string insertion, and Steiner zone reconstruction. In the Steiner point selection operator, a greedy method is used to determine the visiting positions. Furthermore, while the three insertion operators are effective because they consider both the sequence and positions simultaneously, their study revealed that these operators are also very time consuming.

The first exact algorithm for the CETSP was proposed in [6]. The authors formulated a mixed integer programming model (MIP) based on a discretization scheme that provides both lower and upper bounds. However, the quality of the solution depends on the granularity of the discretization scheme, which limits the applicability of this approach. Another exact algorithm [7], based on branch-and-bound and second-order cone programming (SOCP), can reach some optimal solutions in a finite number of steps. These exact algorithms can provide optimal solutions for instances of reasonable size. However, they are not practical for solving large instances.

There are also several discretization-based heuristic methods for solving the CETSP. Carrabs et al. [8] proposed two discretization schemes, namely the perimetral discretization scheme (PDS) and the internal discretization scheme (IDS), to transform the CETSP into a Generalized TSP (GTSP), which is then solved via a mixed integer programming model (MIP). Building on this work, they presented an improved version of the adaptive internal discretization scheme in [9], which combines the discretization scheme with second-order cone programming (SOCP).

In 2020, Carrabs et al. [12] developed a meta-heuristic method called (lb/ub)Alg based on their previous work [9]. The method used an innovative discretization scheme and a carousel greedy algorithm [14] to select appropriate neighborhoods to extend the partial solution until a feasible solution is generated. This method can produce very high-quality solutions in a short amount of time. The discretization scheme is a useful approach for dealing with continuous problems. However, it can impose restrictions on the search space and potentially degrade the quality of the final solution.

Very recently (2022), Di Placido et al. [3] proposed an effective genetic algorithm (GA) for solving the CETSP. They designed a crossover operator, which can be considered as a kind of multi-point crossover, and a diversification-oriented mutation operator. The algorithm also includes three improvement procedures: 2-opt operator, SOCP, and a greedy algorithm called *3Alg*, which was inspired by [11]. GA produced 32 new best solutions out of 62 benchmark instances and matched 27 best-known solutions discovered by other methods. These impressive results highlight the potential of population-based search approaches for solving the CETSP and motivate our work reported in this paper.

In addition to these studies, Yang et al. [10] (2018) addressed the more general TSP with arbitrary neighborhoods (TSPN), which includes the CETSP as a special case. They introduced a double-loop hybrid algorithm (HA), which integrates particle swarm optimization and genetic algorithms. In their work, a linear descending inertia weight particle swarm optimization is used in the outer loop to explore continuous visiting positions, while the inner loop uses a genetic algorithm to optimize the discrete visiting sequence. Given that the HA algorithm is designed for the more general TSPN, the results on the CETSP instances are not competitive with those obtained by recent CETSP algorithms such as SZVNS [11], (lb/ub)Alg [12], and GA [3].

This review shows that the state-of-the-art in solving the CETSP has continuously improved. However, none of the existing methods perform consistently well on the well-known benchmark instances. In what follows, we present a highly effective memetic algorithm to extend our ability to better solve the CETSP.

3. Memetic algorithm for the CETSP

In this section, we introduce the proposed memetic algorithm for solving the CETSP (MA-CETSP). Memetic algorithms [15,16] are a hybrid evolutionary method that combines the genetic search framework with local optimization. This approach has been shown to be highly effective and efficient in solving various combinatorial optimization problems with a permutation representation [17–21], such as the CETSP studied in this paper. Our MA-CETSP algorithm includes three main original components: a dedicated crossover tailored to the CETSP, a powerful VND-based local optimization procedure, and a fitness-and-distance based population management procedure. It also integrates a preprocessing procedure to reduce the input problem and a mutation operator to promote diversity.

3.1. General approach

The CETSP is a complex problem that involves both discrete optimization of finding the shortest tour like the TSP and continuous optimization of finding optimal visiting positions. To handle the complexity of the problem, we use a decomposition approach where we use genetic search to deal with the discrete optimization part, focusing on exploring alternative sequences on a set of fixed points, while we use local optimization to improve the solution by addressing both the discrete and continuous optimization through carefully designed operators.

The proposed algorithm follows the general scheme of memetic algorithms, as shown in Algorithm 1. After preprocessing the input instance (line 4), the algorithm initializes a population of candidate solutions (line 5) and records the current best solution S^* (line 6). The algorithm then explores new candidate solutions by iteratively generating offspring solutions with the crossover operator. In each generation (lines 7–20), two parent solutions are selected at random and recombined to generate an offspring solution (lines 8 and 9). The offspring is first mutated (line 10) to introduce more diversity, and then improved using a VND-based local search (line 11). Finally, it is used to update the population according to the specific update strategy (line 12). During the search, if a new best solution is found, it is recorded as the new best solution S^* (lines 14 and 15); otherwise, the stagnation generation counter $Iter_{st}$ is incremented (lines 16 and 17). The algorithm stops and returns the best solution S^* found (line 21) when the given termination condition is met (line 7), which is either a prefixed maximum number of generations $Iter_{max}$ or a maximum patience ρ value for stagnation generations $Iter_{st}$. In the following sections, we describe the design of each component of the proposed algorithm.

Algorithm 1 Main framework of MA-CETSP

1: **Input**: Instance I, Maximum number of generations $Iter_{max}$, Patience for stagnation generations ρ.
2: **Output**: The best solution S^*.
3: $Iter \leftarrow 0, Iter_{st} \leftarrow 0$ /* current generation and stagnation generation counter */
4: $I \leftarrow Preprocessing(I)$ /* Section 3.2 */
5: $P \leftarrow Initialization(I)$ /* Section 3.3.1 */
6: $S^* \leftarrow BestSolution(P)$ /* record current best solution */
7: **while** $Iter \leq Iter_{max}$ **and** $Iter_{st} \leq \rho$ **do**
8: $S_A, S_B \leftarrow RandomParents(P)$
9: $S' \leftarrow MSX(S_A, S_B)$ /* Section 3.3.2 */
10: $S' \leftarrow Mutation(S')$ /* Section 3.3.3 */
11: $S' \leftarrow VND(S')$ /* Section 3.4 */
12: $P \leftarrow PopulationManagement(S', P)$ /* Section 3.3.4 */
13: $S_{best} \leftarrow BestSolution(P)$ /* the best solution in this generation */
14: **if** $f(S_{best}) < f(S^*)$ **then**
15: $S^* \leftarrow S_{best}$ /* update the best solution */
16: **else**
17: $Iter_{st} \leftarrow Iter_{st} + 1$ /* stagnation counter is incremented */
18: **end if**
19: $Iter = Iter + 1$
20: **end while**
21: **return** S^*

3.2. Preprocessing

The proposed algorithm preprocesses the input instance before the execution of the algorithm to reduce the problem size and improve computational efficiency, like in [3,11].

(1) Remove targets that are dominated by the depot p_0. If the disk neighborhood of a target v_i contains the depot p_0, then v_i will be visited when visiting the depot and can be removed from the list of targets to be visited.
(2) Remove targets that are dominated by other targets. If the disk neighborhood of a target v_i is fully contained within the disk neighborhood of another target v_j with a larger radius, then v_j can be removed because it will be visited when visiting v_i. Note that this condition only applies when targets have different radius disk neighborhoods.

3.3. Genetic search

As mentioned previously, the genetic search in the proposed algorithm primarily addresses the discrete optimization part concerning the visiting sequence of fixed points. It includes population initialization based on K-means, multi-step crossover, mutation, and population management.

3.3.1. K-means based population initialization

Inspired by the work of [22], the proposed algorithm adopts a K-means clustering-based method to initialize the population, as shown in Algorithm 2. Firstly, all targets are grouped into K clusters based on

Algorithm 2 K-means based population initialization

Input: Targets $V = \{v_1, v_2, ..., v_N\}$, Population size N_P.
Output: Population P.
$K \leftarrow \lfloor \sqrt{N} + 0.5 \rfloor$ /* calculate K value */
$C \leftarrow Kmeans(V, K)$ /* generate K clusters */
$P \leftarrow \emptyset$
while $|P| < N_P$ **do**
  $S \leftarrow \emptyset$
  for all $c \in C$ **do**
    $T \leftarrow Permutation(c)$ /* generate a random permutation */
    $S \leftarrow Merge(S, T)$
  end for
  $S \leftarrow GeneratePositions(S)$ /* generate positions for targets in S */
  $P \leftarrow P \cup S$
end while
return P

their positions, where the value of K is calculated based on the number of targets N. Next, a random permutation of targets is generated for each cluster as the sub-visiting sequence. Then, all clusters are connected randomly to form a feasible visiting sequence. Finally, the exact visiting position p_i for every target v_i is determined randomly in the polar coordinate system $p_i = (x_i, y_i) = (R_i \cdot \cos\Theta_i, R_i \cdot \sin\Theta_i)$, where the angle Θ_i is generated in the range $[0, 2\pi]$ and the radius R_i is generated in the range $[0, r_i]$ with r_i being the disk neighborhood radius of target v_i. This initialization method helps to avoid excessively long edges in the initial solution and preserve the quality of the solution.

3.3.2. Multi-step crossover

Crossover is a crucial operator in genetic algorithms. To be effective, the crossover needs to be meaningful with respect to the optimization objective and to allow an offspring solution to inherit good features from the parents while maintaining some diversity [23]. Such a crossover helps the algorithm to explore promising new search areas.

Algorithm 3 Multi-step crossover

Input: Parent solutions S_A and S_B.
Output: Offspring solution S.
Generate multigraph $G = (V, E)$ from S_A and S_B
$S \leftarrow \emptyset, C_g \leftarrow \emptyset$ /* offspring solution and giant AB-cycle */
$m \leftarrow RandomInt(N/2) + 1$ /* generate m as step length */
while $\exists e \in E$ is not visited **do**
  $C \leftarrow \emptyset$ /* an empty AB-cycle */
  $S_p \leftarrow RandomParent(S_A, S_B)$ /* choose a parent */
  while C is not completed **do**
    $C \leftarrow MStep(G, S_p, m)$ /* take m steps along the edges in S_p */
    $S_p \leftarrow SwitchParent(S_A, S_B)$ /* switch to the other parent */
  end while
  $T = ExtractSubtour(C)$ /* extract sub-tour from the AB-cycle */
  $S \leftarrow Merge(S, T)$ /* merge T into S */
  $C_g \leftarrow Merge(C_g, C)$ /* merge C into C_g */
end while
$S \leftarrow InheritPositions(S, S_A, S_B)$ /* inherit positions from parents */
return S

For the CETSP, we propose the multi-step crossover (MSX) operator to recombine two parent solutions randomly selected from the population. We use this simple random parent selection technique, because the advanced population management method (see Section 3.3.4) ensures that the solutions of the population are always of high-quality and sufficiently separated from each other.

Given two parent solutions S_A and S_B, one offspring solution is generated through the following steps (see the pseudo-code shown in Algorithm 3 and the example in Fig. 2).

(1) Construct a multigraph $G = (V, E)$ such that V is the set of the N targets and $(i, j) \in E$ if at least one of the parent solutions S_A and S_B goes from a point p_i of target i to a point p_j of target j. In the example of Fig. 2, Fig. 2(a) shows the parent solutions S_A and S_B represented by red and blue lines, respectively, while Fig. 2(b) shows the corresponding multigraph graph G. Note that in the multigraph G, there are two edges between some pairs of targets (e.g., C-D, G-H...), indicating that the parent solutions both go from a point of one target to a point of the other target.
(2) Generate a random value for m (line 5 in Algorithm 3). This value determines the step length in the AB-cycle generation process. m is set to 2 in the examples presented in Figs. 2(c) and 2(d).
(3) Construct the TSP tour (lines 6–16 in Algorithm 3). Start with an empty giant AB-cycle C_g and an empty TSP tour S. Then, iteratively repeat the process of constructing a m-step AB-cycle C and a sub-tour T, merging them with the giant AB-cycle C_g and TSP tour S, respectively, until all edges in E are visited. At this point the complete TSP tour is obtained. Note that additional edges might be introduced during the merging of the sub-tour T with the TSP tour S.

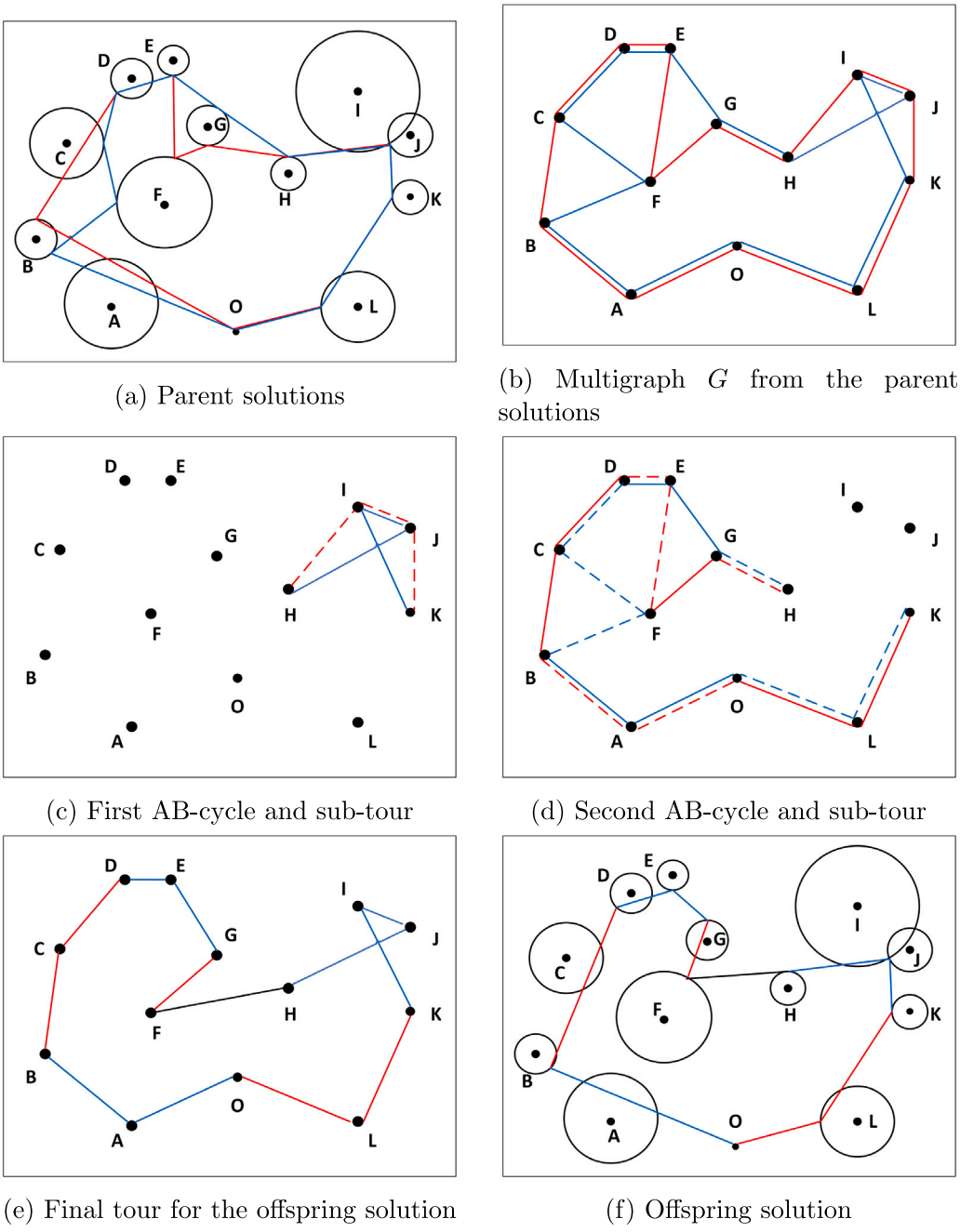
To build a m-step AB-cycle C and its sub-tour T, start from an arbitrary parent solution and a randomly selected vertex. If the current giant AB-cycle C_g is empty, choose an arbitrary vertex; otherwise, select, from the constructed giant AB-cycle, a vertex that has unvisited edges. Then alternately take m edges from each parent solution S_A and S_B until an AB-cycle is completed, which is characterized by (1) returning to the starting vertex, and (2) having no unvisited edges connected to the starting vertex. During the AB-cycle process, if there is no accessible edge in the current parent and the current step count $m_c < m$, the following strategies are applied:

  (a) If $m_c = 0$, borrow one edge from another parent and continue taking m steps in the current parent.
  (b) If $m_c > 0$, stop the exploration in the current parent and switch to the other parent.

From the AB-cycle, a sub-tour T is extracted as a *partial tour* by eliminating the edges according to the order of visit.

We now illustrate this process using the example of Fig. 2. Fig. 2(c) shows the first AB-cycle and its sub-tour. This AB-cycle starts from a random vertex (I) and a random parent (the blue parent). Then with the step length $m = 2$, two blue edges I-J-H are added to the AB-cycle, followed by the addition of two red edges H-I-J from the red parent. At this point, we should switch to the blue parent to continue from vertex J. However, since there is no blue edge connected to J, the strategy (a) is used to borrow a red edge J-K and then continue with the blue edge K-I. The complete AB-cycle I-J-H-I-J-K-I is then obtained as we return to the initial vertex I and I has no unvisited edges. Finally, we eliminate the edges (indicated by dotted lines) based on the order of visit, resulting in the first sub-tour K-I-J-H. This AB-cycle and its sub-tour serve as the first non-empty giant AB-cycle C_g and (partial) TSP tour S, since C_g and S are initialized as empty.

Continuing with the example in Fig. 2(d), another AB-cycle is generated starting from the vertex K, which is a vertex with unvisited edges in the giant AB-cycle. The new AB-cycle traverses the red and blue edges (with $m = 2$) from S_A and S_B alternatively: K-L-O (red), O-A-B (blue), B-C-D (red), D-E-G (blue), G-H



(a) Parent solutions

(b) Multigraph G from the parent solutions

(c) First AB-cycle and sub-tour

(d) Second AB-cycle and sub-tour

(e) Final tour for the offspring solution

(f) Offspring solution

Fig. 2. An illustrative example for the multi-step crossover MSX.

(red), H-G (blue), G-F-E-D (red), D-C-F-B (blue), B-A-O (red), O-L-K (blue). The blue edge H-G and red edge G-F are taken based on the strategy (b), while the red edge E-D and blue edge F-B are taken according to the strategy (a). Dotted lines indicate the elimination of edges based on the order of visit. Notably, edges G-H and H-G are excluded as vertex H has been visited in the first AB-cycle shown in Fig. 2(c). The resulting sub-tour is K-L-O-A-B-C-D-E-G-F. After visiting all edges, the construction of AB-cycles ends, and the final TSP tour (as shown in Fig. 2(e)) is achieved by merging the obtained sub-tour with the current partial TSP tour S and possibly introducing new edges (the black edge F-H in Fig. 2(e)).

(4) Convert the TSP tour to a CETSP tour by inheriting visiting positions of the parent solutions (line 17 in Algorithm 3). Throughout the above steps, every vertex in the offspring tour that represents a target belongs to a corresponding parent solution, so its exact visiting position is inherited from that parent. The final offspring solution is constructed. Fig. 2(f) shows the final offspring solution of our illustrative example.

Our multi-step crossover is inspired by the crossover presented in [3] (referred to as GAX) and the Edge Assembly Crossover (EAX) [24]. GAX is a type of multi-point crossover, which randomly cuts off parent solutions and introduces arbitrary new edges that are not derived from the parents and can be considered generally harmful [25]. In contrast, the MSX adopts the concept of AB-cycles from EAX and constructs the offspring by focusing on edges instead of points. Unlike EAX, which alternates between parent solutions for edge selection, MSX consecutively selects m edges from each parent. Although MSX cannot completely avoid introducing new edges (e.g., the edge F-H in Fig. 2(e)), MSX can better capture the connectivity between the points

in a solution, leading to more meaningful and feasible offspring solutions for our problem. In addition, MSX generates diversified solutions by retaining good features from both parents, allowing the algorithm to explore new search regions, while maintaining a low computational complexity of $O(n)$.

3.3.3. Mutation

Mutation is a useful tool to maintain diversity in the population. The proposed algorithm uses a random *Swap* operator to exchange the order of two visited points in the visiting sequence of the solution. The mutation is applied with the mutation probability $p = Iter_{st}/2\rho$, where ρ is the patience parameter for stagnation generation and $Iter_{st}$ is the number of stagnation generations.

3.3.4. Population management

To maintain the balance between intensification and diversification, the proposed algorithm uses a population management mechanism that carefully controls the diversity of the population [26,27]. First, the graph edit distance d [28] is used to measure the dissimilarity between two solutions. The main idea is to evaluate the unmatched edge ratio. Specifically, the distance is defined as $100\%\times(1-|E_c|/|E|)$, where $|E|$ is the number of edges equal to the number of target points N, and $|E_c|$ represents the number of common edges between two solutions. The distance $d_P(S)$ of a solution S from the population P is then defined as the minimum distance between S and all other solutions in the population, as shown in Eq. (2).

$$d_P(S) = \min_{S_i \in P} d(S, S_i), \qquad S_i \neq S \tag{2}$$

$$\psi(S) = \frac{Rank_{obj}(S) + \lambda * Rank_{dist}(S)}{|P|} \tag{3}$$

The fitness function ψ of Eq. (3) is used to assign a fitness score to each solution of P. The fitness score $\psi(S)$ considers both the rank in terms of the objective value ($Rank_{obj}(S)$) and the distance ($Rank_{dist}(S)$), while λ is a coefficient to adjust the influence of the distance on the fitness score.

Note that the lower the fitness score, the better the fitness of the solution. The fitness score is used to update the population in the proposed algorithm. The population is allowed to expand in $N_P/2$ generations and then it is reduced back to the original size N_P based on the fitness scores.

Finally, the proposed algorithm uses a distance threshold μ to prevent the solutions in the population from becoming too similar. When inserting a new solution into the population, its distance to all existing solutions in the population is calculated, and the solution is only inserted if its distance to the population is no smaller than μ. This strategy ensures a guaranteed diversity of the population and reduces the risk of premature convergence.

3.4. Variable neighborhood descent for local optimization

Local optimization is a crucial component of the proposed algorithm as it intensively explores the search space to find high-quality local optimal solutions. To achieve this, we use Variable Neighborhood Descent (VND), more precisely the pipe VND [29], which sequentially applies a set of local search operators in a pipeline fashion. Specifically, we use sequence-only optimization (LKH solver), position-only optimization (disk-constrained position optimization, SOCP), and joint optimization of sequence and position (*Joint-Relocate*, *Joint-Swap*). Among these local search operators, LKH solver and SOCP are from the literature while disk-constrained position optimization (DCPO) and joint optimization operators are introduced in this work. These operators are executed in a specific order, as outlined in Algorithm 4.

Since the optimization of exact visiting positions in continuous space is computationally complex, we firstly adopt a strategy where we optimize the visiting sequence while fixing the visiting positions and vice versa, which means that we perform position-only optimization and sequence-only optimization alternatively (DCPO - LKH solver - DCPO). It is worth noting that DCPO is executed twice. The first one optimizes the visiting positions of the offspring solution generated by the crossover and serves as a pre-processing step for the LKH solver. The second one optimizes the visiting positions in the sequence provided by the LKH solver and serves as a pre-processing step for the following joint optimization operators. We then execute the joint optimization of sequence and position with *Joint-Relocate* and *Joint-Swap* to further improve the solution. Finally, we obtain the final solution using SOCP with the visiting sequence found before. The designed search operators and search strategy ensure that a thorough and intensive exploitation, resulting in a more refined and accurate solution.

Algorithm 4 Variable neighborhood descent based local search

```
Input: The solution which will be improved S.
Output: The improved solution S'.
S' = DCPO(S)                    /* Section 3.4.3 */
S' = LKH(S')                    /* Section 3.4.1 */
S' = DCPO(S)                    /* Section 3.4.3 */
S' = JointRelocate(S)           /* Section 3.4.4 */
S' = JointSwap(S)               /* Section 3.4.4 */
S' = SOCP(S)                    /* Section 3.4.2 */
return S'
```

3.4.1. LKH solver

The LKH solver [30] is a state-of-the-art TSP solver. We utilize it to optimize the sequence of fixed visiting positions, as in the TSP. As the LKH solver only accepts integer type data, we multiply all coordinate data by 1000 to maintain accuracy.

To further enhance the efficiency of LKH solver, we have designed a strategy to reduce the problem size. Since the *squeeze strategy* employed in Section 3.4.3 may result in an input solution with many redundant points whose coordinates are the same, we removed these redundant points before applying the LKH solver. These removed points are then added back to the optimized solution.

3.4.2. Second-order cone programming

The second-order cone programming (SOCP) [13] was first utilized for the CETSP in [5]. Given a visiting sequence, the SOCP model can be built to find the optimal exact visiting positions in polynomial time. The SOCP model can be formulated as follows:

$$\min \sum d_i \tag{4}$$

subject to

$$dx_i = x_i - x_{i+1} \qquad \forall i \in 0, \dots, N \tag{5}$$

$$dy_i = y_i - y_{i+1} \qquad \forall i \in 0, \dots, N \tag{6}$$

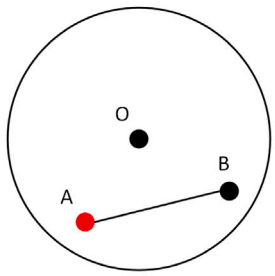
$$dx_i^2 + dy_i^2 \leq d_i^2 \qquad \forall i \in 0, \dots, N \tag{7}$$

$$(\bar{x}_i - x_i)^2 + (\bar{y}_i - y_i)^2 \leq r_i^2 \qquad \forall i \in 0, \dots, N \tag{8}$$

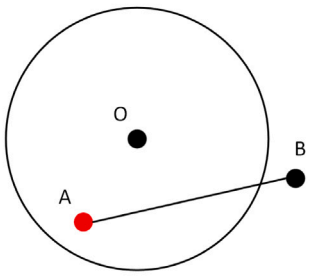
$$d_i \geq 0 \qquad \forall i \in 0, \dots, N \tag{9}$$

$$x_i, y_i, dx_i, dy_i \in \mathbb{R} \qquad \forall i \in 0, \dots, N \tag{10}$$

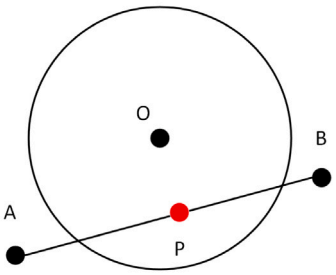
The objective is to minimize the total distance of all connected visiting points, as shown in Formula (4), where d_i represents the Euclidean distance between the connected visiting points p_i and p_{i+1}, where dx_i and dy_i are the decompositions of the distance on the x-axis and y-axis, respectively. They are calculated according to Formulas (5) and (6). It is worth noting that SOCP formulations require constraints with quadratic terms to be expressed as inequalities instead of equalities. Hence, the distance d_i is defined with an inequality in Formula (7). Additionally, Formula (8) constrains the coordinates (x_i, y_i) of visiting point p_i to be within the disk neighborhood of the target v_i with coordinates $(\bar{x}_i, \bar{y}_i)$, where r_i is the radius of the disk.



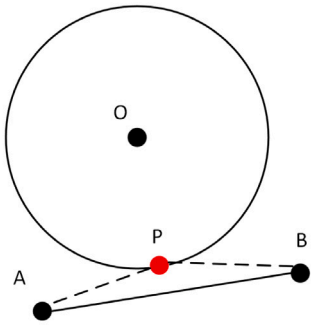
(a) Both A and B are inside the disk.



(b) Either A or B is inside the disk.



(c) Both A and B are outside the disk and line segment AB intersects with the disk boundary.



(d) Both A and B are outside the disk and line segment AB does not intersect with the disk boundary.

Fig. 3. Cases in disk-constrained position optimization.

3.4.3. Disk-constrained position optimization

SOCP is an effective approach to find the optimal positions in polynomial time for a given visiting sequence of points. However, it is computationally expensive. Therefore, we propose a disk-constrained position optimization (DCPO) based on the greedy algorithm in [11] as a replacement for the SOCP before and after the LKH solver to optimize the visiting positions. This approach can significantly reduce the computation time while still providing a good-quality solution.

The core idea of the approach is based on the geometric relationship between circles and points. Given any three consecutive visiting points A, P, B, let O be the center of the disk where P is located, we can fix A and B to find the best position of P satisfying the constraints. There are several cases to consider (as shown in Fig. 3).

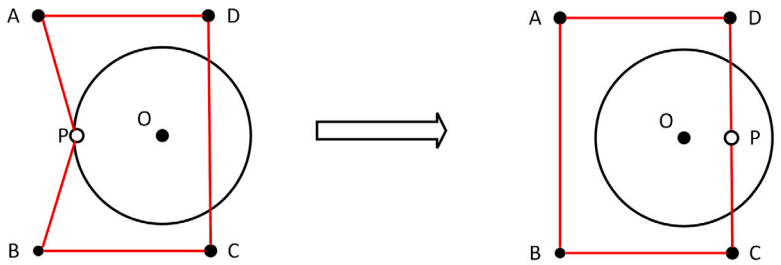
If both A and B are inside the disk O, any point in the line segment AB can be selected as P. We take a *squeeze strategy* here, which means we prefer that more points can be integrated together. Thus, we choose A as P (Fig. 3(a)). If only one of them is inside the disk, we choose the one inside (Fig. 3(b)).

If both A and B are outside the disk O, there are two subcases. The first is that the line segment AB intersects the disk O, in which case we take the only intersection point or the midpoint of the two intersection points as P (Fig. 3(c)). The second is when the line segment AB does not intersect the disk O, and the problem becomes the well-known mathematical problem of Alhazen's problem [31]. The objective is to find the point P on the circumference of the disk O such that the sum $AO+BO$ is minimized (Fig. 3(d)). To solve this problem, we use a binary search algorithm to obtain an approximate position. We represent the sum $AO + BO$ as a function $\xi = d(A, O) + d(B, O)$, where $d(*)$ returns the Euclidean distance between two points. As the function ξ changes periodically as the point P moves around the circumference of O, we approximate the position of P using the binary search algorithm so that the derivative of the function $\xi' = 0$ in the interval where the function ξ first decreases and then increases.

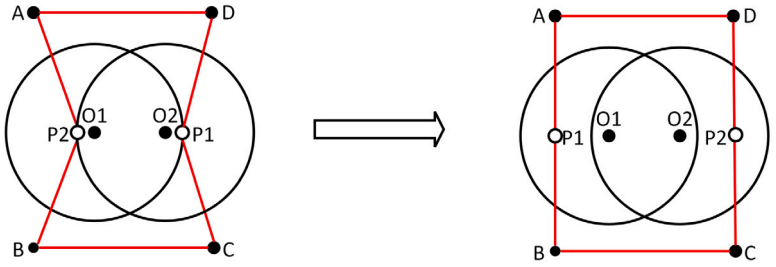
3.4.4. Joint-relocate and joint-swap

The above operators only consider either the visiting sequence or the visiting position. To further improve the solution, we design the operators *Joint-Relocate* and *Joint-Swap* that take into account both the sequence and position. The *Joint-Relocate* operator removes a point from the sequence and inserts it into a better position, as illustrated in Fig. 4(a), while the *Joint-Swap* operator exchanges the visiting order of two disconnected points, as shown in Fig. 4(b). Both of them will simultaneously update positions of the moved points. They employ a best-position search strategy for every point within κ-nearest-neighbors that is a kind of granular search [32], which means the best move in the κ-nearest-neighbors will be taken for every point and the search repeats until no improvement can be found. Additionally, the κ-nearest-neighbors are dynamically updated based on the historical visiting positions for each target.

These two operators take the position into account by recalculating the exact position of a point according to the principles of the DCPO presented in Section 3.4.3. However, the binary search algorithm used



(a) Joint-Relocate



(b) Joint-Swap

Fig. 4. Joint optimization operators.

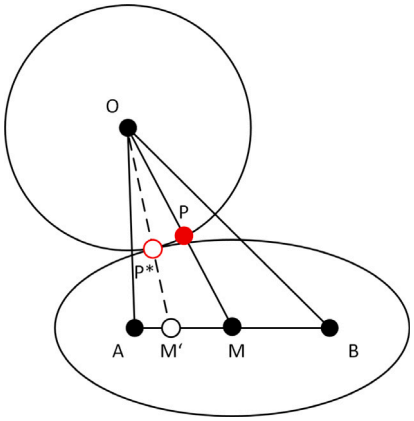


Fig. 5. Approximate strategy for joint optimization.

in the case of Alhazen's problem [31] for searching a huge neighborhood is of high complexity. To address this, we propose an alternative strategy.

It can be proved geometrically that the optimal point P^* is the point of tangency between the disk O and an ellipse with focus at A and B, which is tangent to the disk O. Therefore, according to the features of ellipse, as shown in Fig. 5, the straight line OM' is the bisector of $\angle AP^*B$, which can be approximated using the midline OM of $\angle AOB$. We take the intersection point of the midline OM with the circumference of the disk O as P. With this strategy, the move evaluation can be done in constant time and greatly improves computational efficiency.

4. Computational results

To evaluate the proposed algorithm, we test it on benchmark instances and compare it with the state-of-the-art algorithms in the literature.

4.1. Benchmark instances

The proposed algorithm was evaluated on the set of 62 popular benchmark instances from [5] with different sizes from 10 to 1000 targets[1]. These instances were obtained from three sources: *TSBLIB* [33], *Teams*, and *Geometric*.

- The *TSBLIB* subset consists of 28 instances generated from 7 graphs (*d493, dsj1000, kroD100, lin318, pcb442, rat195,* and *rd400*) with sizes ranging from 100 to 1000 nodes. These graphs comprise two with uniformly generated nodes (*d493, dsj1000*), two with clustered nodes (*kroD100, lin318*), two with drill-press problems (*pcb442, rat195*), and one with nodes arranged in rough lines (*rd400*).
- The *Teams* subset consists of 14 instances from 7 graphs whose names contain *team* or *bonus* discussed in [1], including three graphs with uniformly generated points and four graphs with clustered points.
- The remaining 20 instances belong to the *Geometric* subset and are from [5]. This set includes 9 *Bubbles* instances where the nodes are arranged in overlapping concentric squares, 5 *ConcentricCircles* instances where the nodes are arranged in concentric circles, 5 *RotatingDialonds* instances where the nodes are arranged in non-overlapping concentric squares, and one *chaoSingleDep* instance based on the problem given in [34].

[1] The benchmark instances are available at https://drum.lib.umd.edu/handle/1903/9822.

Table 2
Parameters tuning results.

Parameters	Data type	Candidate values	Best configuration
N_P	categorical	{10, 20, 30, 40, 50}	20
$Iter_{max}$	categorical	{1000, 2000, 3000, 4000, 5000}	5000
λ	real	(0, 1)	0.96
μ	integer	(0, 10)	5
κ	categorical	{10, 20, 30, 40, 50}	50

Overlap ratio (OR) is a metric introduced in [5] to reflect the characteristics of CETSP instances and to estimate the potential improvement over the TSP solution. In this study, we employ the OR calculation method proposed in [3]. For a given CETSP instance, let L represent the length of the smallest side of the rectangle that encloses all the targets, the overlap ratio is calculated as the ratio between the average value of the targets' radii and L, i.e. $OR = \frac{\sum_{i=1}^{N} r_i}{N \times L}$. Based on their characteristics of overlap ratios and disk radii, these 62 benchmark instances are categorized into three groups.

- The first group (G1) consists of 27 instances with different overlap ratios (OR), where the radii of targets are identical in an instance but the overlap ratios differ among instances.
- The second group (G2) comprises 21 instances with fixed overlap ratios, where the radii of targets are identical in an instance as well, while the overlap ratios are fixed and equal to 2% (G2_0.02), 10% (G2_0.1), and 30% (G2_0.3).
- The third group (G3) comprises 14 instances with arbitrary radii, where targets have different radius.

We will present our computational results based on this categorization.

4.2. Experimental protocol and parameters tuning

The proposed algorithm was implemented in C++ and the experiments were conducted on a computer equipped with an AMD-Opteron-4184 2.8 GHz processor and 4 GB RAM, running Linux. Gurobi 10 [35] was utilized to solve the second-order cone programming (SOCP) within the proposed algorithm.

The algorithm incorporates five parameters: population size N_P, maximum number of generations $Iter_{max}$, coefficient of the fitness function λ, distance threshold μ, size of nearest neighbors κ. To determine a suitable setting for these parameters, we employed the automatic parameter tuning tool IRACE [36]. 9 instances from different groups were randomly selected as training instances, and the tuning budget was set to 1000.

Table 2 provides the results of parameters tuning, including the data type, candidate values, and the best configuration for each parameter identified by IRACE. For our experiments, we consistently used the values of the parameters shown in Table 2. Additionally, considering the correlation between the patience for stagnation generations ρ and the maximum number of generations $Iter_{max}$, we experimentally set $\rho = Iter_{max}/10$. We make sure that the stopping criterion used in our algorithm is comparable to that of the main reference algorithm GA [3]. We performed 20 independent runs for each instance.

4.3. Reference algorithms

For our comparative study, we use as reference methods the best-known solutions (BKS) ever reported in the literature and three best-performing state-of-the-art algorithms: SZVNS (2019) [11], which uses variable neighborhood search to explore search operators similar to those used in our algorithm, (lb/ub)Alg (2020) [12], which still holds several BKS, and the most recent GA (2022) [3], which holds the BKS for most benchmark instances.

In [11], SZVNS was compared with HA [10] and Mennell's heuristics proposed in [5] using the 62 benchmark instances presented in Section 4.1 and the published results in [5,10]. The results showed that SZVNS generally outperformed HA and Mennell's heuristics, and demonstrated its ability to produce excellent results in a remarkably short running time.

In [12], (lb/ub)Alg used the same 62 benchmark instances to present a comparative study with SZVNS [11], HA [10], and some selected Mennell's heuristics from [5]. Comparing the published results of these reference algorithms, the results showed the superior overall performance of (lb/ub)Alg over HA and Mennell's heuristics in both objective values and running time. Furthermore, (lb/ub)Alg demonstrated competitive competence with SZVNS, albeit with slightly slower execution.

For the latest GA [3], it was compared with HA [10], SZVNS [11], and (lb/ub)Alg [12] on the 62 benchmark instances presented in Section 4.1. The comparative study was also based on the results published in the literature, while mentioning the CPU models used by the compared algorithms. The experiments showed that GA completely dominates the reference algorithms, especially HA, in terms of objective values, although it is generally slower in terms of running time.

From these previous studies, it is clear that SZVNS [11], (lb/ub)Alg, [12] and GA [3] dominate other CETSP algorithms in the literature, hold together the current best-known results for the 62 benchmark instances, and represent the state of the art for solving the CETSP. As a result, we use them as our reference methods in comparative study. Moreover, to ensure a meaningful comparative study, we follow the practice for comparative studies adopted in [3,11,12]. That is, we test our algorithm on the same set of 62 benchmark instances presented in Section 4.1 and use the published results as our reference objective values.

The experimental environments used by the reference algorithms are as follows. SZVNS [11] was programmed in C++ and executed on a laptop with a 2.6 GHz processor and 4 GB RAM. (lb/ub)Alg [12] was programmed in Java and run on a machine with an Intel i5 2.3 GHz processor and 8 GB RAM. GA [3] was programmed in Java and run on a Windows 10 Home machine with an AMD Ryzen 73750H 2.3 GHz processor and 16 GB RAM.

To account for processor differences, we introduce a time conversion ratio γ to harmonize the timing information. Finally, to facilitate future experimental comparisons, we are taking the groundbreaking step of releasing the source code of our algorithm to the community,[2] filling the current gap that there is no publicly available code for the CETSP.

Table 3 shows the processor information for the computers used to run our algorithm and the reference algorithms. The time conversion ratio γ (last column) is computed as the ratio of CPU scores obtained from PassMark,[3] a reputable CPU benchmarking platform, comparing the processor used in our experiments with the processors used by the reference algorithms. Given the lack of exact processor models for the (lb/ub)Alg and SZVNS, we use equivalent processors based on processor information and the dates when their papers were submitted for publication.

4.4. Computational results and comparisons

We present a summary of the computational results for the different groups of benchmark instances and provide a general overview of the performance of our MA-CETSP algorithm. We consider the best results

[2] The source code of our MA-CETSP algorithm and our solutions for the benchmark instances are available at https://github.com/leizy1008/MA-CETSP.

[3] https://www.passmark.com/

Table 3
Processor information for MA-CETSP, GA [3], (lb/ub)Alg [12], and SZVNS [11] and time conversion ratio γ.

Algorithm	Processor	Base frequency	Launch date	CPU mark	γ
MA-CETSP	AMD Opteron 4184	2.80 GHz	Q3 2011	2872	**1.00**
GA	AMD Ryzen 7 3750H	2.30 GHz	Q2 2019	8210	**2.86**
(lb/ub)Alg	Intel Core i5-8259U	2.30 GHz	Q2 2018	7979	2.78
	Intel Core i5-8300H	2.30 GHz	Q2 2018	7473	2.60
	Intel Core i5-8600T	2.30 GHz	Q2 2018	9304	3.24
	Intel Core i5-7360U	2.30 GHz	Q1 2017	3895	1.36
	Intel Core i5-6350HQ	2.30 GHz	Q1 2016	4255	1.48
	Avg			6581	**2.29**
SZVNS	Intel Core i5-7300U	2.60 GHz	Q1 2017	3677	1.28
	Intel Core i7-6770HQ	2.60 GHz	Q1 2016	7111	2.48
	Intel Core i5-6440HQ	2.60 GHz	Q3 2015	5094	1.77
	Intel Core i7-6600U	2.60 GHz	Q3 2015	3457	1.20
	Intel Core i7-6700HQ	2.60 GHz	Q3 2015	6525	2.27
	Intel Core i7-5700EQ	2.60 GHz	Q2 2015	5803	2.02
	Intel Core i7-4720HQ	2.60 GHz	Q1 2015	5752	2.00
	Intel Core i7-5600U	2.60 GHz	Q1 2015	3022	1.05
	Avg			5055	**1.76**

Table 4
Summary of comparative results of MA-CETSP against BKS, GA [3], (lb/ub)Alg [12], and SZVNS [11].

Group		G1	G2_0.02	G2_0.1	G2_0.3	G3	Total
#Instances		27	7	7	7	14	62
#Optima		9	0	4	7	3	23
MA-CETSP vs BKS	#Wins	**13**	**6**	**3**	0	**8**	**30**
	#Ties	14	1	4	7	6	32
	#Losses	0	0	0	0	0	0
	p-value	–	–	–	–	–	7.98E−08
MA-CETSP vs GA	#Wins	**13**	**6**	**6**	0	**8**	**33**
	#Ties	14	1	1	7	6	29
	#Losses	0	0	0	0	0	0
	p-value	–	–	–	–	–	1.32E−08
MA-CETSP vs (lb/ub)Alg	#Wins	**17**	**7**	**5**	**1**	**12**	**42**
	#Ties	10	0	2	6	2	20
	#Losses	0	0	0	0	0	0
	p-value	–	–	–	–	–	8.75E−10
MA-CETSP vs SZVNS	#Wins	**21**	**6**	**7**	**5**	**10**	**49**
	#Ties	6	1	0	2	4	13
	#Losses	0	0	0	0	0	0
	p-value	–	–	–	–	–	2.36E−11

among 20 runs. The summary is presented in Table 4. The line *#Instances* indicates the number of instances in the corresponding group, and the line *#Optima* shows the number of instances whose optimal solutions are known. The lines *#Wins*, *#Ties*, and *#Losses* respectively indicate the number of instances where MA-CETSP achieves better, same, and worse results compared to the references. Furthermore, to confirm the statistical difference in the results, we conducted a Wilcoxon signed-rank test [37] with a confidence level of 0.05, and the corresponding *p-values* are shown in the table.

Table 4 clearly shows that the MA-CETSP algorithm provides comparable or better results compared to BKS and all reference algorithms. MA-CETSP obtained all 23 known optimal values and 30 new best upper bounds out of the remaining 39 instances. The *p-values* ($\ll 0.05$) indicates that the proposed algorithm statistically performs better than the reference algorithms.

Tables 5–7 show detailed comparative results on the three groups of instances between MA-CETSP and other heuristic approaches including GA [3], (lb/ub)Alg [12], and SZVNS [11]. In each table, column *Instance* shows the names of the instances. Note that the instances with $*$ indicate that their optimal solutions are known and given by [7]. Columns *N* and *OR* respectively represent the target size and overlap ratio of the corresponding instance. Column *BKS* indicates the best-known solutions in the literature. Column f shows the best objective value achieved by each algorithm. Column *Time* reports the total running time in seconds for each algorithm. For the reference algorithms, column $\gamma \times Time$ reports the converted time with the time conversion ratio γ given in Table 3. Column *Gap* shows the percentage gap between the best value obtained by each algorithm and the *BKS* for that instance, which is calculated according to the following formula: $100\% \times \frac{f-BKS}{BKS}$. Thus a negative gap indicates an improved best solution.

For the proposed algorithm, we report both the best and average values obtained over 20 independent runs. We also report the running time in seconds to reach the best solution in column t and average objective value with standard deviation in column $f \pm sd$. Finally, it should be noted that due to the differences in the programming languages and computers used by the compared algorithms, the timing information is provided only for indicative purposes.

Table 5 presents the results for the 27 instances of Group 1 with different overlap ratios, where each instance has the targets with same radius but there are different overlap ratios among them. Our MA-CETSP algorithm yields the best results for all instances, by achieving the BKS in the literature and improving the remaining instances. Notably, significant improvements are obtained for *bubbles7* (2.01% gap), *bubbles8* (3.33% improvement), *bubbles9* (4.91% improvement), and *team4_400* (2.28% improvement). The overall average values are also competitive, with 9 out of 27 instances improving the *BKS*. The algorithm achieves these results in a short computation time. Furthermore, the standard deviations for almost all instances are very low, with the exception of *bubbles9*, which is the only instance with a standard deviation over 10. The high standard deviation value of

Table 5
Comparative results between MA-CETSP, BKS, GA [3], (lb/ub)Alg [12], and SZVNS [11] for the 27 instances of Group G1 with different overlap ratios and identical radii.

Instance	N	OR	BKS	MA-CETSP (Best)				MA-CETSP (Avg)				GA				(lb/ub)Alg				SZVNS			
				f	t	Time	Gap	$f \pm sd$	t	Time	Gap	f	Time	γ×Time	Gap	f	Time	γ×Time	Gap	f	Time	γ×Time	Gap
bonus1000	1000	12.26%	387.13	**384.37**	1792.93	2016.81	−0.71%	390.40 ± 4.10	1841.73	2020.71	0.85%	411.85	8786.91	25 130.56	6.39%	387.13	116.63	267.08	0.00%	403.06	1109.67	1953.02	4.11%
bubbles1*	36	11.11%	349.14	349.14	0.63	31.12	0.00%	349.14 ± 0.00	0.87	33.83	0.00%	349.14	2.21	6.32	0.00%	349.14	11.42	26.15	0.00%	349.14	0.24	0.42	0.00%
bubbles2*	76	9.09%	428.28	428.28	1.35	41.73	0.00%	428.28 ± 0.00	1.85	41.97	0.00%	428.28	2.22	6.35	0.00%	428.28	22.68	51.94	0.00%	428.28	1.09	1.92	0.00%
bubbles3*	126	7.69%	529.96	529.96	134.83	193.23	0.00%	529.96 ± 0.00	57.68	113.00	0.00%	529.96	7.29	20.85	0.00%	529.96	130.17	298.09	0.00%	532.21	2.51	4.42	0.42%
bubbles4	184	6.67%	805.46	**802.97**	107.45	173.63	−0.31%	**805.37 ± 1.70**	141.61	209.72	−0.01%	805.46	27.84	79.62	0.00%	805.56	130.11	297.95	0.01%	825.33	9.53	16.77	2.47%
bubbles5	250	5.88%	1038.16	**1035.32**	145.38	247.52	−0.27%	**1037.36 ± 1.21**	198.61	288.07	−0.08%	1038.16	45.07	128.90	0.00%	1061.64	116.63	267.08	2.26%	1073.43	37.37	65.77	3.40%
bubbles6	324	5.26%	1229.66	**1220.07**	243.81	358.74	−0.78%	**1227.15 ± 3.83**	252.68	368.82	−0.20%	1229.66	176.16	503.82	0.00%	1313.02	217.20	497.39	6.78%	1263.68	14.00	24.64	2.77%
bubbles7	406	4.76%	1607.31	**1575.04**	511.48	650.29	−2.01%	**1586.78 ± 9.35**	534.21	681.41	−1.28%	1607.31	809.15	2314.17	0.00%	1650.04	304.46	697.21	2.66%	1639.33	50.29	88.51	1.99%
bubbles8	496	4.35%	1946.72	**1881.93**	600.38	790.10	−3.33%	**1895.74 ± 7.97**	681.67	877.10	−2.62%	1946.72	1230.27	3518.57	0.00%	2021.27	416.30	953.33	3.83%	1972.99	110.30	194.13	1.35%
bubbles9	594	4.00%	2259.22	**2148.40**	644.57	893.73	−4.91%	**2168.21 ± 10.63**	783.96	1041.72	−4.03%	2259.22	2655.50	7594.73	0.00%	2413.31	296.14	678.16	6.82%	2330.31	168.18	296.00	3.15%
chaoSingleDep*	200	2.84%	1039.61	1039.61	11.16	76.44	0.00%	1039.61 ± 0.00	19.64	90.96	0.00%	1039.61	10.69	30.57	0.00%	1039.61	89.69	205.39	0.00%	1039.63	12.67	22.30	0.00%
concentricCircles1	16	15.00%	53.16	53.16	0.82	29.31	0.00%	53.16 ± 0.00	1.65	32.41	0.00%	53.16	0.94	2.69	0.00%	53.16	6.43	14.72	0.00%	53.16	0.10	0.18	0.00%
concentricCircles2	36	7.50%	153.13	153.13	105.10	139.32	0.00%	153.22 ± 0.33	47.64	82.35	0.06%	153.13	2.77	7.92	0.00%	154.81	51.96	118.99	1.10%	154.88	0.27	0.48	1.14%
concentricCircles3	60	5.00%	270.04	**270.01**	25.71	67.01	−0.01%	270.04 ± 0.06	54.49	97.77	0.00%	270.04	2.79	7.98	0.00%	272.69	362.24	829.53	0.98%	272.49	1.28	2.25	0.91%
concentricCircles4	104	3.75%	452.64	**451.87**	140.10	193.29	−0.17%	453.09 ± 0.91	145.10	199.59	0.10%	452.64	12.56	35.92	0.00%	466.52	98.86	226.39	3.07%	461.36	5.82	10.24	1.93%
concentricCircles5	148	3.00%	632.99	**632.98**	142.54	208.92	0.00%	633.43 ± 0.55	124.10	188.63	0.07%	632.99	26.04	74.47	0.00%	659.36	351.21	804.27	4.17%	647.84	10.06	17.71	2.35%
rotatingDiamonds1*	20	20.00%	32.39	32.39	0.64	33.70	0.00%	32.39 ± 0.00	0.75	32.74	0.00%	32.39	0.80	2.29	0.00%	32.39	7.06	16.17	0.00%	32.39	0.12	0.21	0.00%
rotatingDiamonds2*	60	5.00%	140.48	140.48	1.38	46.81	0.00%	140.48 ± 0.00	3.33	46.65	0.00%	140.48	1.94	5.55	0.00%	140.48	176.13	403.34	0.00%	140.48	0.54	0.95	0.00%
rotatingDiamonds3	180	3.33%	380.88	380.88	15.58	87.69	0.00%	380.88 ± 0.00	22.40	98.40	0.00%	380.88	10.60	30.32	0.00%	380.89	615.47	1409.43	0.00%	380.89	8.81	15.51	0.00%
rotatingDiamonds4	320	1.43%	770.66	770.66	100.03	229.92	0.00%	770.67 ± 0.02	62.96	200.87	0.00%	770.66	41.85	119.69	0.00%	772.00	271.20	621.05	0.17%	770.68	18.49	32.54	0.00%
rotatingDiamonds5	680	1.11%	1510.75	1510.75	164.34	597.20	0.00%	1510.77 ± 0.03	399.78	862.71	0.00%	1510.75	376.41	1076.53	0.00%	1531.74	210.99	483.17	1.39%	1510.88	53.62	94.37	0.01%
team1_100*	100	9.33%	307.34	307.34	10.32	54.29	0.00%	307.34 ± 0.00	13.37	58.31	0.00%	307.34	3.86	11.04	0.00%	307.34	78.86	180.59	0.00%	307.34	3.20	5.63	0.00%
team2_200*	200	20.06%	246.68	246.68	120.87	181.62	0.00%	246.68 ± 0.00	59.94	115.46	0.00%	246.68	23.75	67.93	0.00%	246.68	36.06	82.58	0.00%	246.69	16.21	28.53	0.00%
team3_300	300	7.02%	464.2	**461.89**	343.45	435.69	−0.50%	**462.16 ± 0.58**	313.96	405.34	−0.44%	464.20	99.30	284.00	0.00%	476.43	56.88	130.26	2.63%	465.80	31.48	55.40	0.34%
team4_400	400	5.01%	685.52	**669.91**	522.60	648.08	−2.28%	**675.46 ± 3.04**	543.24	669.68	−1.47%	685.52	428.43	1225.31	0.00%	702.69	120.25	275.37	2.50%	698.05	43.47	76.51	1.83%
team5_499	499	2.00%	700.5	**693.80**	369.70	561.82	−0.96%	**697.37 ± 1.68**	596.33	772.86	−0.45%	700.50	995.23	2846.36	0.00%	708.45	828.33	1896.88	1.13%	703.38	88.17	155.18	0.41%
team6_500*	500	27.06%	225.22	225.22	100.27	208.22	0.00%	225.22 ± 0.00	98.53	208.37	0.00%	225.22	518.12	1481.82	0.00%	225.22	8.15	18.66	0.00%	226.18	719.32	1266.00	0.43%
Mean	–	–	690.64	681.34	235.46	340.60	−0.60%	684.09 ± 3.41	259.34	364.42	−0.35%	691.55	603.66	1726.45	0.24%	708.51	190.06	435.23	1.46%	701.11	93.22	164.06	1.07%

Table 6
Comparative results between MA-CETSP, BKS, GA [3], (lb/ub)Alg [12], and SZVNS [11] for the 21 instances of Group G2 with fixed overlap ratios and identical radii.

Instance	N	OR	BKS	MA-CETSP (Best)				MA-CETSP (Avg)				GA				(lb/ub)Alg				SZVNS			
				f	t	Time	Gap	$f \pm sd$	t	Time	Gap	f	Time	$\gamma\times$Time	Gap	f	Time	$\gamma\times$Time	Gap	f	Time	$\gamma\times$Time	Gap
Overlap ratio 2%																							
d493	492	2.00%	202.23	**199.02**	509.47	689.44	−1.59%	**199.67 ± 0.37**	576.47	748.29	−1.26%	202.23	864.47	2472.38	0.00%	205.39	247.04	565.72	1.56%	205.74	69.72	122.71	1.74%
dsj1000	999	2.00%	938.71	**909.50**	2175.15	2578.76	−3.11%	**914.21 ± 2.05**	1857.10	2252.60	−2.61%	938.71	20 325.36	58 130.53	0.00%	955.57	998.99	2287.69	1.80%	943.83	151.77	267.12	0.55%
kroD100	99	2.00%	159.04	159.04	104.90	158.32	0.00%	159.11 ± 0.14	48.56	103.16	0.04%	159.04	9.05	25.88	0.00%	160.09	167.14	382.75	0.66%	159.04	9.34	16.44	0.00%
lin318	317	2.00%	2838.54	**2816.58**	212.79	327.54	−0.77%	**2826.16 ± 5.76**	267.89	380.49	−0.44%	2838.54	174.28	498.44	0.00%	2902.53	292.90	670.74	2.25%	2842.32	31.17	54.86	0.13%
pcb442	441	2.00%	322.54	**319.85**	1153.16	1334.95	−0.84%	**321.05 ± 0.97**	733.22	917.77	−0.46%	322.54	826.50	2363.79	0.00%	377.51	805.44	1844.46	17.04%	325.02	52.88	93.07	0.77%
rat195	194	2.00%	158.32	**157.97**	232.82	322.37	−0.22%	158.85 ± 0.58	193.23	282.07	0.33%	158.32	55.23	157.96	0.00%	166.51	569.55	1304.27	5.17%	160.06	11.16	19.64	1.10%
rd400	399	2.00%	1032.04	**1018.46**	321.90	450.63	−1.32%	**1021.19 ± 1.79**	510.00	671.91	−1.05%	1032.04	686.27	1962.73	0.00%	1085.75	662.60	1517.35	5.20%	1039.77	27.70	48.75	0.75%
Overlap ratio 10%																							
d493*	492	10.00%	100.72	100.72	224.86	352.52	0.00%	101.12 ± 0.72	395.68	515.35	0.40%	101.31	345.37	987.76	0.59%	100.72	33.87	77.56	0.00%	102.92	142.94	251.57	2.18%
dsj1000	999	10.00%	374.06	**373.76**	739.12	942.49	−0.08%	374.52 ± 1.17	491.21	698.80	0.12%	376.87	3951.05	11 300.00	0.75%	374.06	83.00	190.07	0.00%	393.06	356.79	627.95	5.08%
kroD100*	99	10.00%	89.67	89.67	10.67	50.76	0.00%	89.68 ± 0.06	16.14	59.40	0.01%	89.67	3.92	11.21	0.00%	89.67	193.44	442.98	0.00%	89.92	1.99	3.50	0.28%
lin318*	317	10.00%	1394.63	1394.63	109.67	193.10	0.00%	1399.96 ± 5.55	163.18	248.39	0.38%	1404.89	74.77	213.84	0.74%	1405.07	26.50	60.69	0.75%	1414.66	65.98	116.12	1.44%
pcb442	441	10.00%	145.82	**142.57**	356.87	467.38	−2.23%	**144.30 ± 2.07**	428.68	541.56	−1.04%	145.82	466.16	1333.22	0.00%	146.03	224.95	515.14	0.14%	152.73	54.99	96.78	4.74%
rat195*	194	10.00%	67.99	67.99	111.73	170.90	0.00%	67.99 ± 0.00	100.50	161.42	0.00%	68.14	21.31	60.95	0.22%	68.14	95.19	217.99	0.22%	68.32	9.47	16.67	0.49%
rd400	399	10.00%	458.41	**452.83**	435.65	540.53	−1.22%	**453.96 ± 0.99**	446.81	555.74	−0.97%	458.41	612.93	1752.98	0.00%	460.21	101.27	231.91	0.39%	474.78	96.98	170.68	3.57%
Overlap ratio 30%																							
d493*	492	30.00%	69.76	69.76	15.88	116.74	0.00%	69.76 ± 0.00	25.66	128.59	0.00%	69.76	233.54	667.92	0.00%	69.79	2.72	6.23	0.04%	69.90	58.07	102.20	0.20%
dsj1000*	999	30.00%	199.95	199.95	103.59	249.61	0.00%	199.95 ± 0.00	118.00	260.48	0.00%	199.95	1181.44	3378.92	0.00%	199.95	6.89	15.78	0.00%	203.07	316.41	556.88	1.56%
kroD100*	99	30.00%	58.54	58.54	1.43	39.25	0.00%	58.54 ± 0.00	1.80	39.70	0.00%	58.54	2.20	6.29	0.00%	58.54	0.37	0.85	0.00%	58.54	4.58	8.06	0.00%
lin318*	317	30.00%	765.96	765.96	10.14	71.47	0.00%	765.96 ± 0.00	15.27	79.57	0.00%	765.96	29.69	84.91	0.00%	765.96	1.21	2.77	0.00%	766.16	49.01	86.26	0.03%
pcb442*	441	30.00%	83.54	83.54	25.02	117.86	0.00%	83.54 ± 0.00	35.68	127.00	0.00%	83.54	121.23	346.72	0.00%	83.54	0.58	1.33	0.00%	83.80	196.54	345.91	0.31%
rat195*	194	30.00%	45.7	45.70	2.75	50.19	0.00%	45.70 ± 0.00	4.37	53.99	0.00%	45.70	6.49	18.56	0.00%	45.70	0.33	0.76	0.00%	45.70	19.01	33.46	0.00%
rd400*	399	30.00%	224.84	224.84	16.26	91.67	0.00%	224.84 ± 0.00	39.37	116.51	0.00%	224.84	67.12	191.96	0.00%	224.84	4.19	9.60	0.00%	224.98	74.86	131.75	0.06%
Mean	–	–	463.38	459.57	327.33	443.64	−0.54%	460.96 ± 1.95	308.04	425.85	−0.31%	464.04	1431.35	4093.67	0.11%	473.60	215.15	492.70	1.68%	467.82	85.78	150.97	1.19%

Table 7
Comparative results between MA-CETSP, BKS, GA [3], (lb/ub)Alg [12], and SZVNS [11] for the 14 instances of Group 3 with different overlap ratios and arbitrary radii.

Instance	N	OR	BKS	MA-VND (Best)				MA-VND (Avg)				GA				(lb/ub)Alg				SZVNS			
				f	t	Time	Gap	$f \pm sd$	t	Time	Gap	f	Time	$\gamma\times$Time	Gap	f	Time	$\gamma\times$Time	Gap	f	Time	$\gamma\times$Time	Gap
bonus1000rdmRad	1000	6.13%	932.07	**917.61**	741.85	890.66	−1.55%	**921.26 ± 3.14**	351.93	488.28	−1.16%	932.07	480.23	1373.46	0.00%	955.41	728.55	1668.38	2.50%	938.27	3.93	6.92	0.67%
d493rdmRad	492	13.44%	134.28	**134.23**	19.86	71.85	−0.04%	**134.23 ± 0.00**	53.53	106.54	−0.04%	134.28	5.58	15.96	0.00%	134.74	93.34	213.75	0.34%	135.02	0.26	0.46	0.55%
dsj1000rdmRad	999	12.47%	624.75	**624.60**	109.13	176.72	−0.02%	625.00 ± 0.42	98.41	167.34	0.04%	624.75	25.86	73.96	0.00%	625.92	286.23	655.47	0.19%	625.25	2.73	4.80	0.08%
kroD100rdmRad	99	4.03%	141.83	141.83	105.05	158.19	0.00%	141.88 ± 0.24	48.47	96.26	0.04%	141.83	6.97	19.93	0.00%	142.39	46.25	105.91	0.39%	141.83	87.16	153.40	0.00%
lin318rdmRad	317	10.05%	2047.42	**2047.11**	12.10	66.44	−0.02%	**2047.11 ± 0.00**	22.00	77.66	−0.02%	2047.42	8.71	24.91	0.00%	2055.77	59.52	136.30	0.41%	2082.25	3.18	5.60	1.70%
pcb442rdmRad	441	9.39%	220	**219.22**	105.21	160.56	−0.36%	**219.58 ± 0.36**	103.76	163.42	−0.19%	220.00	16.16	46.22	0.00%	220.44	153.38	351.24	0.20%	221.16	1.36	2.39	0.53%
rat195rdmRad*	194	42.60%	68.22	68.22	0.76	32.17	0.00%	68.22 ± 0.00	1.07	34.95	0.00%	68.22	1.35	3.86	0.00%	68.22	9.97	22.83	0.00%	68.22	10.57	18.60	0.00%
rd400rdmRad	399	0.99%	1246.69	**1238.28**	442.44	635.69	−0.67%	**1240.89 ± 1.32**	442.08	645.05	−0.47%	1246.69	701.06	2005.03	0.00%	1305.46	1110.35	2542.70	4.71%	1257.73	0.67	1.18	0.89%
team1_100rdmRad*	100	7.69%	388.54	388.54	1.17	38.02	0.00%	388.54 ± 0.00	2.27	40.46	0.00%	388.54	1.84	5.26	0.00%	388.54	20.80	47.63	0.00%	388.54	3.89	6.85	0.00%
team2_200rdmRad	200	5.19%	613.66	613.66	33.72	93.86	0.00%	614.28 ± 1.28	61.76	123.82	0.10%	613.66	16.04	45.87	0.00%	616.82	209.44	479.62	0.51%	626.90	0.93	1.64	2.16%
team3_300rdmRad*	300	23.70%	378.09	378.09	10.22	48.46	0.00%	378.09 ± 0.00	9.35	49.82	0.00%	378.09	3.75	10.73	0.00%	378.51	16.65	38.13	0.11%	379.84	34.51	60.74	0.46%
team4_400rdmRad	400	2.05%	1000.31	**984.24**	601.63	734.98	−1.61%	**986.44 ± 2.23**	381.55	518.11	−1.39%	1000.31	317.04	906.73	0.00%	1025.76	658.03	1506.89	2.54%	1006.71	2.62	4.61	0.64%
team5_499rdmRad	499	20.14%	446.19	446.19	10.21	51.36	0.00%	446.19 ± 0.00	12.06	55.94	0.00%	446.19	2.52	7.21	0.00%	446.51	51.43	117.77	0.07%	446.19	5.05	8.89	0.00%
team6_500rdmRad	500	10.02%	620.98	**620.89**	187.92	261.21	−0.02%	**620.94 ± 0.13**	92.96	159.83	−0.01%	620.98	26.15	74.79	0.00%	626.18	347.08	794.81	0.84%	621.99	19.77	34.80	0.16%
Mean	–	–	633.07	630.19	170.09	244.30	−0.31%	630.90 ± 1.15	120.09	194.82	−0.22%	633.07	115.23	329.57	0.00%	642.19	270.79	620.10	0.92%	638.56	12.62	22.20	0.56%

this instance can be attributed to its large number of targets (594) and low overlap ratio value (OR = 4%). Table 6 displays the results for the 21 instances of Group 2 with different overlap ratios, generated from 7 *TSPLIB* instances with low (2%), moderate (10%), and high (30%) overlap ratios respectively. Our MA-CETSP algorithm achieves 9 new best upper bounds, with 3.11% improvement for *dsj1000* (OR = 2%) and 2.23% improvement for *pcb442* (OR = 10%). Moreover, we observe that a high overlap ratio makes the problem easier to solve. When the overlap ratio approaches 0, the problem reduces to the standard TSP. All instances with an overlap ratio of 30% can be solved to optimality in a short time, while instances with an overlap ratio of 2% are more challenging and require longer running time. Table 7 presents the results for the 14 instances of Group 3 with arbitrary radii, where each instance contains targets with different radii. Our MA-CETSP algorithm finds new best upper bounds for 8 instances, including the challenging *bonus1000rmdRad* (1.55% improvement) and *team4_400rdmRad* (1.61% improvement).

Finally, we performed additional experiments in terms of running time to compare MA-CETSP with each reference algorithm, including GA, (lb/ub)Alg, and SZVNS, to provide a comprehensive evaluation of the overall performance of the proposed algorithm. In these experiments, we used, as our termination time, the running time reported in the literature for each reference algorithm and for each instance, which is a more stringent termination condition considering their better processor performance as shown in Table 3. We ran our algorithm 20 times to solve each instance and report the best results. The results are presented in Table 8. In this table, column f reports the objective value of MA-CETSP, while f_{ref} presents the objective value of the reference algorithms. Column *Gap* is calculated using the formula $100\% \times \frac{f-f_{ref}}{f_{ref}}$. Column *Time* recalls the given termination time, which is also the reported running time of the reference algorithms. The last two rows present the average statistical data and the *#Wins/Ties/Losses*, indicating the number of instances where MA-CETSP achieved better, equal, and worse results compared to the references. It can be seen that, in spite of the stringent termination conditions, the proposed algorithm shows a consistently competitive performance.

In summary, the extensive experiments conducted confirm that MA-CETSP performs remarkably well on these three sets of benchmark instances, demonstrating its ability to find high quality solutions.

5. A real-world case

The CETSP has a number of practical applications. In order to demonstrate the practical applicability of the proposed algorithm in real-world scenarios, we present a practical case study, which concerns laser welding robot path planning, as discussed in [4]. Spot welding robots are widely used in various industries such as machinery, aerospace, and automobile manufacturing. An effective path planning algorithm for welding robot can significantly enhance productivity and reduce production costs.

In the context of the industrialized car door welding problem, the robot's task is to weld a set of target points to secure the car door within an allowed range of deviation. Clearly, this problem can be viewed as a CETSP by treating the welding targets, along with their allowed range of deviation, as the targets with disk neighborhoods in the CETSP, we can then apply CETSP techniques to find an optimized feasible welding path for the robot.

In [4], a two-phase method was proposed to tackle this problem. The first phase aims to find a visiting sequence of the welding targets, by solving a proposed integer programming model, while the second phase determines the exact positions based on this sequence through a formulated nonlinear programming model. Two strategies were proposed for the first phase. One involves solving the integer programming model, which can be considered as a kind of Generalized TSP, by using 4 middle points of 4 quarters on the perimeter for each target disk (referred to as 2-phase-GTSP), and the other is to solve directly a TSP integer programming model by only considering the center points (referred to as 2-phase-TSP).

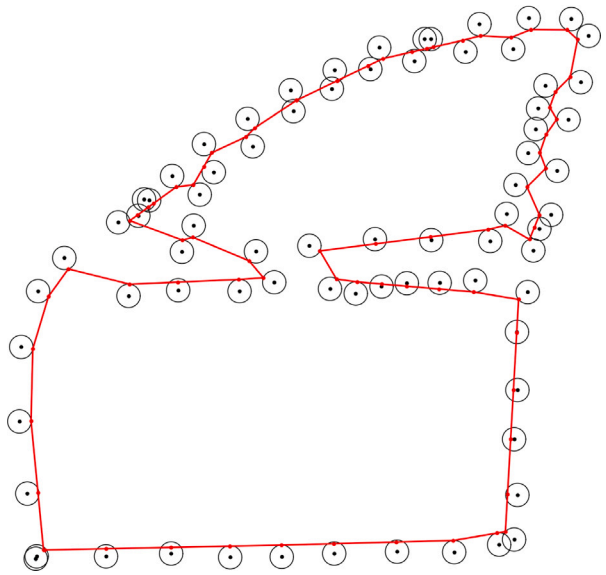


Fig. 6. Example of car door welding problem with 75 welding points and a radius of 30.

We conducted experiments on the largest car door welding instance presented in [4], which contains 75 welding points. To diversify the scenarios, we generated six additional instances[4] based on the same target points provided by [4] with different disk radii: 25, 30, 35, 40, 45, and 50. The configurations for the experiments aligned with those detailed in Section 4, except for the maximum number of generations $Iter_{max}$, which was set to 500 due to the modest scale of these instances. We ran our algorithm 10 times to solve each instance. Additionally, we implemented the two-phase methods 2-phase-GTSP and 2-phase-TSP [4] as our comparison references. As the two-phase methods are deterministic methods, they were executed once with a maximum running time set to 3600 s, utilizing the Gurobi solver [35]. All experiments were carried out in the same environment as described in Section 4.

Table 9 shows the comparative results, with the columns retaining the same meaning as in Tables 5–7, except for the column *Gap*, which uses the objective value of MA-CETSP (best) (denoted as f_{best}) as a reference. Specifically, it is calculated as $100\% \times \frac{f-f_{best}}{f_{best}}$. According to results of Table 9, the proposed algorithm consistently outperforms the reference methods, by finding better solutions within significantly shorter running times. This characteristic underscores its practicality and effectiveness in real-world scenarios. Fig. 6 shows a solution generated by our algorithm for the car door welding instance with a radius of 30. The planned robot route is depicted by the red lines, while the red points indicate the exact welding locations within each designated welding area.

This case study shows that the proposed algorithm is able to effectively solve the welding robot path planning problem in the car industry and sheds light on its potential for other practical applications.

6. Performance analysis

In this section, we investigate, through a series of experiments and comparative analyses, the impacts of the algorithmic components on the performance of the proposed algorithm. All experiments in this section follow the same experimental protocol given in Section 4.2.

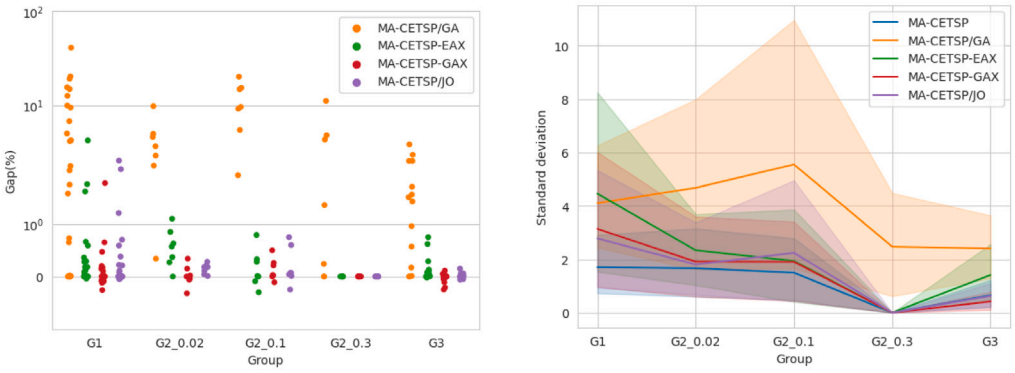
[4] All instances and our solutions are available at https://github.com/leizy1008/MA-CETSP.

Table 8
Comparative results of experiments with termination time for GA [3] vs MA-CETSP, (lb/ub)Alg [12] vs MA-CETSP, and SZVNS [11] vs MA-CETSP for all 62 instances.

Instance	GA vs MA-CETSP				(lb/ub)Alg vs MA-CETSP				SZVNS vs MA-CETSP			
	f_{ref}	f	Time	Gap	f_{ref}	f	Time	Gap	f_{ref}	f	Time	Gap
bonus1000	411.85	**382.87**	8786.91	−7.04%	387.13	451.10	116.63	16.52%	403.06	**385.93**	1109.67	−4.25%
bubbles1	349.14	349.14	2.21	0.00%	349.14	349.14	11.42	0.00%	349.14	349.14	0.24	0.00%
bubbles2	428.28	428.28	2.22	0.00%	428.28	428.28	22.68	0.00%	428.28	428.28	1.09	0.00%
bubbles3	529.96	532.67	7.29	0.51%	529.96	529.96	130.17	0.00%	532.21	559.47	2.51	5.12%
bubbles4	805.46	806.55	27.84	0.14%	805.56	**803.07**	130.11	−0.31%	825.33	**825.06**	9.53	−0.03%
bubbles5	1038.16	1048.30	45.07	0.98%	1061.64	**1036.02**	116.63	−2.41%	1073.43	**1047.41**	37.37	−2.42%
bubbles6	1229.66	**1222.95**	176.16	−0.55%	1313.02	**1221.75**	217.20	−6.95%	1263.68	1378.72	14.00	9.10%
bubbles7	1607.31	**1577.94**	809.15	−1.83%	1650.04	**1580.87**	304.46	−4.19%	1639.33	1690.83	50.29	3.14%
bubbles8	1946.72	**1883.89**	1230.27	−3.23%	2021.27	**1889.28**	416.30	−6.53%	1972.99	2016.04	110.30	2.18%
bubbles9	2259.22	**2149.17**	2655.50	−4.87%	2413.31	**2244.81**	296.14	−6.98%	2330.31	2357.59	168.18	1.17%
chaoSingleDep	1039.61	1039.61	10.69	0.00%	1039.61	1039.61	89.69	0.00%	1039.63	1039.61	12.67	0.00%
concentricCircles1	53.16	53.16	0.94	0.00%	53.16	53.16	6.43	0.00%	53.16	53.16	0.10	0.00%
concentricCircles2	153.13	155.67	2.77	1.66%	154.81	**153.13**	51.96	−1.08%	154.88	158.23	0.27	2.16%
concentricCircles3	270.04	272.13	2.79	0.77%	272.69	**270.01**	362.24	−0.98%	272.49	**272.32**	1.28	−0.06%
concentricCircles4	452.64	455.99	12.56	0.74%	466.52	**452.17**	98.86	−3.08%	461.36	**459.69**	5.82	−0.36%
concentricCircles5	632.99	636.90	26.04	0.62%	659.36	**632.98**	351.21	−4.00%	647.84	**645.19**	10.06	−0.41%
rotatingDiamonds1	32.39	32.39	0.80	0.00%	32.39	32.39	7.06	0.00%	32.39	32.39	0.12	0.00%
rotatingDiamonds2	140.48	140.48	1.94	0.00%	140.48	140.48	176.13	0.00%	140.48	140.48	0.54	0.00%
rotatingDiamonds3	380.88	380.88	10.60	0.00%	380.89	380.88	615.47	0.00%	380.89	380.88	8.81	0.00%
rotatingDiamonds4	770.66	770.66	41.85	0.00%	772.00	**770.66**	271.20	−0.17%	770.68	770.68	18.49	0.00%
rotatingDiamonds5	1510.75	1510.75	376.41	0.00%	1531.74	**1510.75**	210.99	−1.37%	1510.88	1511.08	53.62	0.01%
team1_100	307.34	307.41	3.86	0.02%	307.34	307.34	78.86	0.00%	307.34	308.72	3.20	0.45%
team2_200	246.68	246.92	23.75	0.10%	246.68	246.72	36.06	0.02%	246.69	247.85	16.21	0.47%
team3_300	464.20	**462.67**	99.30	−0.33%	476.43	**463.33**	56.88	−2.75%	465.80	468.24	31.48	0.52%
team4_400	685.52	**669.83**	428.43	−2.29%	702.69	**678.77**	120.25	−3.40%	698.05	**697.76**	43.47	−0.04%
team5_499	700.50	**693.81**	995.23	−0.96%	708.45	**694.30**	828.33	−2.00%	703.38	**701.02**	88.17	−0.33%
team6_500	225.22	225.22	518.12	0.00%	225.22	261.06	8.15	15.91%	226.18	**225.22**	719.32	−0.43%
d493_or2	202.23	**199.24**	864.47	−1.48%	205.39	**199.61**	247.04	−2.81%	205.74	**202.18**	69.72	−1.73%
dsj1000_or2	938.71	**911.07**	20 325.36	−2.94%	955.57	**913.15**	998.99	−4.44%	943.83	963.25	151.77	2.06%
kroD100_or2	159.04	159.13	9.05	0.06%	160.09	**159.04**	167.14	−0.66%	159.04	159.26	9.34	0.14%
lin318_or2	2838.54	**2817.50**	174.28	−0.74%	2902.53	**2816.99**	292.90	−2.95%	2842.32	**2826.32**	31.17	−0.56%
pcb442_or2	322.54	**320.08**	826.50	−0.76%	377.51	**319.89**	805.44	−15.26%	325.02	**322.92**	52.88	−0.64%
rat195_or2	158.32	159.05	55.23	0.46%	166.51	**157.97**	569.55	−5.13%	160.06	162.02	11.16	1.23%
rd400_or2	1032.04	**1019.21**	686.27	−1.24%	1085.75	**1019.42**	662.60	−6.11%	1039.77	**1029.15**	27.70	−1.02%
d493_or10	101.31	**100.72**	345.37	−0.58%	100.72	105.05	33.87	4.30%	102.92	**100.89**	142.94	−1.98%
dsj1000_or10	376.87	**373.76**	3951.05	−0.82%	374.06	381.63	83.00	2.02%	393.06	**373.78**	356.79	−4.90%
kroD100_or10	89.67	90.00	3.92	0.37%	89.67	89.67	193.44	0.00%	89.92	90.05	1.99	0.14%
lin318_or10	1404.89	**1397.32**	74.77	−0.54%	1405.07	1413.75	26.50	0.62%	1414.66	**1400.83**	65.98	−0.98%
pcb442_or10	145.82	**142.70**	466.16	−2.14%	146.03	**143.27**	224.95	−1.89%	152.73	**147.16**	54.99	−3.64%
rat195_or10	68.14	68.52	21.31	0.56%	68.14	**67.99**	95.19	−0.22%	68.32	70.27	9.47	2.85%
rd400_or10	458.41	**452.81**	612.93	−1.22%	460.21	**455.43**	101.27	−1.04%	474.78	**458.29**	96.98	−3.47%
d493_or30	69.76	69.76	233.54	0.00%	69.79	69.99	2.72	0.29%	69.90	**69.76**	58.07	−0.20%
dsj1000_or30	199.95	199.95	1181.44	0.00%	199.95	212.42	6.89	6.24%	203.07	**199.95**	316.41	−1.54%
kroD100_or30	58.54	58.54	2.20	0.00%	58.54	58.54	0.37	0.00%	58.54	58.54	4.58	0.00%
lin318_or30	765.96	765.96	29.69	0.00%	765.96	771.90	1.21	0.77%	766.16	**765.96**	49.01	−0.03%
pcb442_or30	83.54	83.54	121.23	0.00%	83.54	90.53	0.58	8.36%	83.80	**83.54**	196.54	−0.31%
rat195_or30	45.70	45.70	6.49	0.00%	45.70	45.70	0.33	0.00%	45.70	45.70	19.01	0.00%
rd400_or30	224.84	224.84	67.12	0.00%	224.84	247.61	4.19	10.13%	224.98	**224.84**	74.86	−0.06%
bonus1000rdmRad	932.07	**916.38**	480.23	−1.68%	955.41	**916.38**	728.55	−4.09%	938.27	962.03	3.93	2.53%
d493rdmRad	134.28	134.74	5.58	0.34%	134.74	**134.23**	93.34	−0.38%	135.02	**134.74**	0.26	−0.21%
dsj1000rdmRad	624.75	625.52	25.86	0.12%	625.92	**624.60**	286.23	−0.21%	625.25	634.68	2.73	1.51%
kroD100rdmRad	141.83	142.19	6.97	0.25%	142.39	**141.83**	46.25	−0.39%	141.83	141.83	87.16	0.00%
lin318rdmRad	2047.42	2047.87	8.71	0.02%	2055.77	**2047.11**	59.52	−0.42%	2082.25	**2052.48**	3.18	−1.43%
pcb442rdmRad	220.00	220.11	16.16	0.05%	220.44	**219.22**	153.38	−0.55%	221.16	222.38	1.36	0.55%
rat195rdmRad	68.22	68.22	1.35	0.00%	68.22	68.22	9.97	0.00%	68.22	68.22	10.57	0.00%
rd400rdmRad	1246.69	**1238.07**	701.06	−0.69%	1305.46	**1238.02**	1110.35	−5.17%	1257.73	**1255.48**	0.67	−0.18%
team1_100rdmRad	388.54	388.54	1.84	0.00%	388.54	388.54	20.80	0.00%	388.54	388.54	3.89	0.00%
team2_200rdmRad	613.66	613.78	16.04	0.02%	616.82	**613.66**	209.44	−0.51%	626.90	627.06	0.93	0.03%
team3_300rdmRad	378.09	378.09	3.75	0.00%	378.51	**378.09**	16.65	−0.11%	379.84	**378.09**	34.51	−0.46%
team4_400rdmRad	1000.31	**983.38**	317.04	−1.69%	1025.76	**983.63**	658.03	−4.11%	1006.71	1008.28	2.62	0.16%
team5_499rdmRad	446.19	446.19	2.52	0.00%	446.51	**446.19**	51.43	−0.07%	446.19	446.19	5.05	0.00%
team6_500rdmRad	620.98	620.99	26.15	0.00%	626.18	**620.89**	347.08	−0.85%	621.99	**621.27**	19.77	−0.12%
Mean	601.29	595.48	773.72	−0.48%	613.97	599.71	216.79	−0.62%	607.97	610.43	72.50	0.06%
#Wins/Ties/Losses	21/22/19				37/14/11				28/14/20			

Table 9
Comparative results between MA-CETSP, 2-phase-TSP and 2-phase-GTSP [4] for the car door welding instances with different radii.

Instance	MA-CETSP (Best)			MA-CETSP (Avg)				2-phase-TSP			2-phase-GTSP		
	f	t	Time	$f \pm sd$	t	Time	Gap	f	Time	Gap	f	Time	Gap
car_door_25	5339.75	8.22	15.77	5339.76 ± 0.00	5.18	11.44	0.00%	5405.55	46.79	1.23%	5448.32	3600.00	2.03%
car_door_30	5204.78	10.05	16.15	5204.78 ± 0.00	6.72	12.97	0.00%	5272.49	39.65	1.30%	5436.60	3600.00	4.45%
car_door_35	5073.63	10.08	16.06	5087.02 ± 9.25	8.39	14.47	0.26%	5154.14	45.50	1.59%	5225.03	3600.00	2.98%
car_door_40	4963.66	15.72	22.03	4972.68 ± 14.55	8.21	14.71	0.18%	5043.87	39.65	1.62%	5031.55	3600.00	1.37%
car_door_45	4869.81	10.47	16.23	4881.41 ± 15.34	9.71	15.92	0.24%	4942.81	43.99	1.50%	4992.41	3600.00	2.52%
car_door_50	4778.91	11.28	18.30	4785.45 ± 6.07	11.61	17.76	0.14%	4859.60	41.03	1.69%	4814.74	3600.00	0.75%
Mean	5038.42	10.97	17.42	5045.18 ± 7.53	8.30	14.55	0.14%	5113.08	42.77	1.49%	5158.11	3600.00	2.35%



(a) Gap of average values (b) Standard deviation values

Fig. 7. Comparison of average solution quality and standard deviation.

6.1. Impacts of genetic search, crossover and joint optimization

We first study the impact of the genetic framework, crossover operator, and joint optimization on solution quality and robustness of the algorithm. We modify certain components of the MA-CETSP algorithm to create the following algorithm variants:

(1) MA-CETSP/GA: In this variant, we eliminate the genetic search framework and retain only the VND local search. Specifically, we start with an initial solution generated using the method described in Section 3.3.1, and then optimize the solution using the same local search methods described in Section 3.4. The mutation operator introduced in Section 3.3.3 is used as a perturbation operator to help the algorithm to escape local optima.
(2) MA-CETSP-EAX and MA-CETSP-GAX: In these two variants, we replace our MSX crossover described in Section 3.3.2 with the well-known EAX crossover [24], which has been proven highly effective for the TSP and many TSP variants, and the GAX crossover used in [3], which obtained excellent results for the CETSP respectively. Note that for EAX, the exact visiting position of any target is randomly inherited from two parents.
(3) MA-CETSP/JO: In this variant, we eliminate the joint optimization operators (*Joint-Relocate* and *Joint-Swap*) to assess their impacts on the performance of the algorithm.

Fig. 7 presents a visualized comparison of the results obtained by MA-CETSP and these variants. Fig. 7(a) illustrates the gap, expressed as a percentage, between the average values of the variants and MA-CETSP for every instance: $100\% \times \frac{\overline{f}_{variant} - \overline{f}_{baseline}}{\overline{f}_{baseline}}$. Each data point represents the average gap value of an instance. Fig. 7(b) shows the mean standard deviation values (represented by color lines) with 0.95 confidence intervals (represented by corresponding color blocks) for different groups.

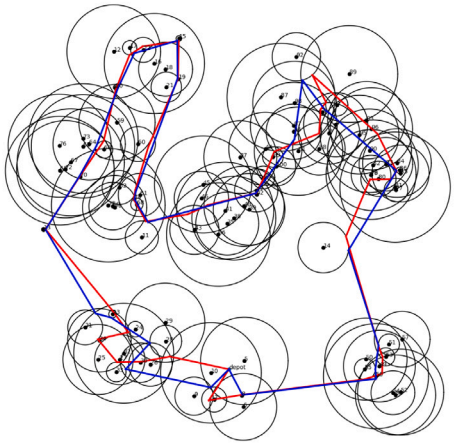
The variant without the genetic framework (MA-CETSP/GA) exhibits the worst performance, with large gaps and standard deviations for most instances. This confirms the significance of the genetic framework for the MA-CETSP algorithm, which includes the crossover, mutation, and population management strategies. On the other hand, the other three variants (MA-CETSP-EAX, MA-CETSP-GAX, and MA-CETSP/JO) show a similar performance. Notably, for some groups (e.g., G2_0.3) that are relatively easy to solve, these variants perform well with low gaps and low standard deviations. However, they show a poorer performance in more challenging groups, particularly in G1, where they exhibit larger gaps compared to MA-CETSP. The results suggest that the genetic framework, along with the specific crossover operator and joint optimization operators, play vital roles for the proposed algorithm in achieving better performance and robustness in solving the problem instances.

6.2. Understanding the multi-step crossover

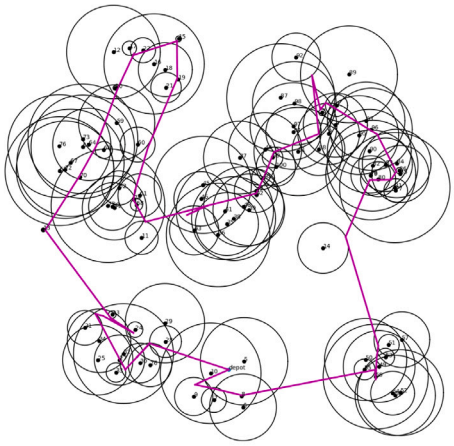
To gain further insights into the crossovers, in particular the multi-step crossover MSX designed for the CETSP, we present an in-depth analysis of the crossover process. Using the instance *Team1_100rmdRad* as an example, Fig. 8 shows two parents S_A and S_B where Figs. 8(b) to 8(d) depict the offspring solutions generated by MSX, EAX, and GAX, respectively. In each figure, we indicate the objective value of the offspring S_i, $i = 1, 2, 3$ and its improved offspring S_i', $i = 1, 2, 3$ from the local optimization VND, and the distance between the offspring and each parent solution.

Upon examination, it becomes apparent that the offspring solution S_2 generated by EAX is highly similar to the parent solution S_A, with a distance of only 3.92. This can be attributed to the characteristics of the CETSP, where the arrangement of points within small local ranges does not significantly affect the solution quality due to the clustering of points within the disk neighborhood areas. Consequently, EAX, which tends to construct AB-cycles [24] within these local ranges, limiting its ability to effectively explore the search space.

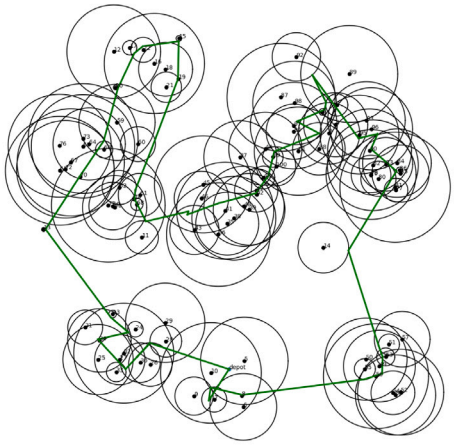
On the other hand, GAX inherits variable-length sub-sequences from the parent solutions, which may contain high-quality sequences and points identified by local search. By randomly splitting the parents,



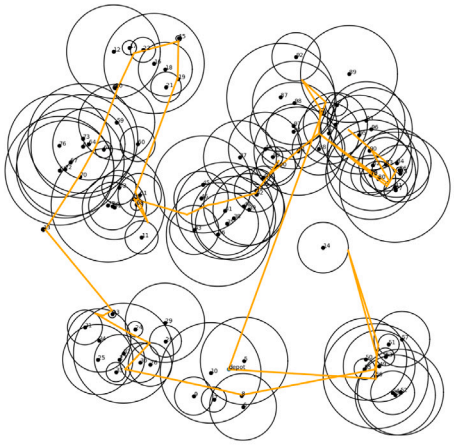
(a) Parent solutions S_A (red) and S_B (blue).
$f(S_A) = 417.70$; $f(S_B) = 393.33$



(b) Offspring S_1 from MSX. $f(S_1) = 462.15$;
$d(S_1, S_A) = 19.60$; $d(S_1, S_B) = 27.45$;
let $S_1' = VND(S_1)$, $f(S_1') = 391.62$



(c) Offspring S_2 from EAX. $f(S_2) = 444.90$;
$d(S_2, S_A) = 3.92$; $d(S_2, S_B) = 43.14$;
let $S_2' = VND(S_2)$, $f(S_2') = 395.01$



(d) Offspring S_3 from GAX. $f(S_3) = 599.07$;
$d(S_3, S_A) = 52.94$; $d(S_3, S_B) = 45.10$;
let $S_3' = VND(S_3)$, $f(S_3') = 395.85$

Fig. 8. Comparison of crossovers on the instance *Team1_100rmdRad*.

as illustrated in Fig. 8(d), GAX introduces diversification. This occasionally leads to very good solutions, as evident in instances with a negative gap in Fig. 7(a). However, the introduction of harmful edges, as depicted in Fig. 8(d), is also more likely. As a result, MA-CETSP-GAX exhibits greater variability and is less stable, as shown in Fig. 7(b).

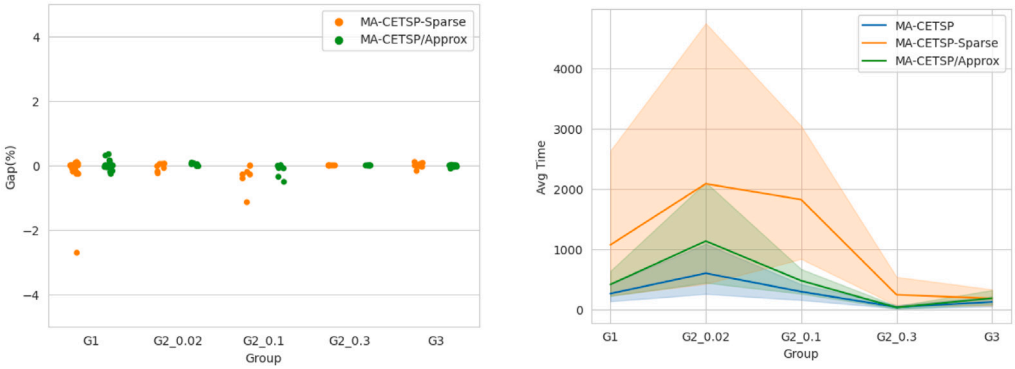
In contrast, the proposed MSX crossover designed for the CETSP combines the strengths of both EAX and GAX. It inherits high-quality sub-sequences from both parents and ensures the consecutive arrangement of edges when switching parents, resulting in high-quality and robust offspring solutions. As shown in Fig. 8(b), the offspring S_1 from MSX maintains a good balance between quality and diversity. After undergoing the local optimization of VND, a better solution is obtained compared to S_2 and S_3, confirming its promise.

These findings provide an explanation for the superior performance of the multi-step crossover, which exhibits low variability across all instance groups (MA-CETSP in Fig. 7(b)), highlighting its stability and consistency.

6.3. Impact on computation time

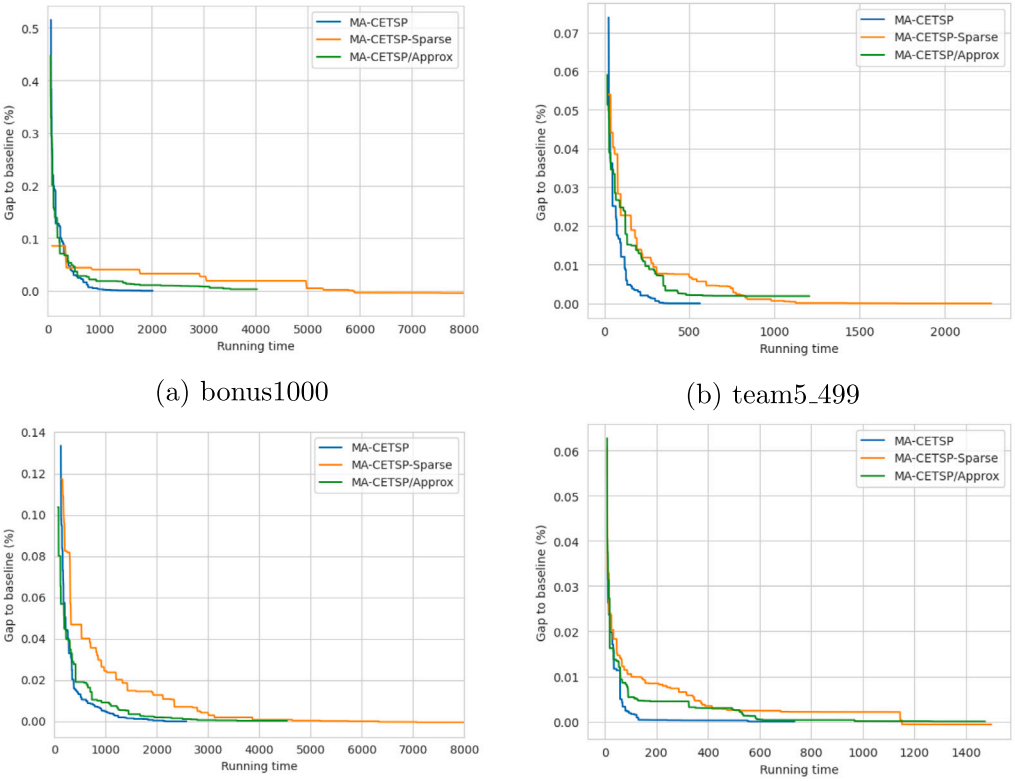
In the proposed algorithm, two important strategies are utilized to enhance computational effectiveness: the *squeeze strategy* in DCPO (Section 3.4.3) and the approximation strategy in joint optimization (Section 3.4.4). In this section, we compare the MA-CETSP algorithm with the following variants to investigate the impact of these two strategies:

(1) MA-CETSP-Sparse: In this variant, we modify DCPO by adopting the *sparse strategy* instead of the *squeeze strategy*. In the *sparse strategy*, the points are expected to be detached. Therefore, when both points A and B are inside the disk O (the case shown in Fig. 3(a)), we take the midpoint as P.
(2) MA-CETSP/Approx: In this variant, we discard the approximation strategy and use binary section search to solve the Alhazen's problem [31] instead (the case shown in Fig. 3(d)).



(a) Gap of average value (b) Average computational time

Fig. 9. Comparison of average solution quality and the computational time.



(a) bonus1000 (b) team5_499

(c) dsj1000 with OR=2% (d) team4_400rdmRad

Fig. 10. Convergence profiles of four instances.

Fig. 9 shows the comparative results between MA-CETSP and these two algorithmic variants. Like Fig. 7(a), Fig. 9(a) shows the average gaps of different variants compared to the MA-CETSP algorithm for every instance. Fig. 9(b) shows the average computation time for the compared algorithms. The results indicate that MA-CETSP-Sparse and MA-CETSP/Approx can achieve better results for certain instances, but for most instances, their performance is comparable to MA-CETSP. However, it is important to note that these variants come with a significantly higher computational cost. The computation time of MA-CETSP with the *sparse strategy* (MA-CETSP-Sparse) is prohibitively long and not acceptable. Similarly, MA-CETSP without the approximation strategy (MA-CETSP/Approx) also requires a long running time, particularly for Group G2_0.02 (low overlap ratio 2%), where the instances are closer to the standard TSP and more difficult to solve. In addition, Fig. 10 shows the best convergence profiles of four representative instances in different groups (*bonus1000*, *team5_499*, *dsj1000* with OR=2% and *team_400rdmRad*) among 20 runs. The results clearly show that MA-CETSP exhibits a faster convergence to high-quality solutions compared to the two variants.

In summary, while the MA-CETSP-Sparse and MA-CETSP/Approx variants yield better results for some instances, their computational requirements make them less practical. The MA-CETSP algorithm with the *squeeze strategy* in DCPO and the approximation strategy in joint optimization strikes a good balance between solution quality and computational efficiency.

7. Conclusion

In this study, we proposed an effective memetic algorithm for the Close-Enough Traveling Salesman Problem. The algorithm incorporates the original multi-step crossover operator considering problem features and an effective local optimization procedure exploring both existing techniques and two new joint optimization operators. The algorithm additionally uses a mutation and distance-and-fitness based population management mechanism to ensure appropriate population diversity, as well as specific strategies to speed up the search. Extensive experiments have been conducted to evaluate the performance of the algorithm in terms of solution quality and computational efficiency.

The results demonstrated that the proposed algorithm outperforms the existing algorithms on the 62 benchmark instances. It successfully reached all 23 known optimal solutions and achieved new best upper bounds for 30 instances out of the remaining 39 instances with for unknown optimal solution. Comparative analysis of different configurations further highlighted the effectiveness and robustness of the proposed algorithm, providing insights into the contributions of individual components. Additionally, the algorithm was successfully applied to a real-world case, showcasing its practical applicability in solving real-world problems modeled as CETSP. The source code of the algorithm, which we make available online, will facilitate such applications.

For future work, there are several potential directions to consider. One avenue is the improvement of the proposed algorithm through the integration of learning techniques, which has shown promise in the field of combinatorial optimization [38]. Furthermore, the algorithm can be adapted and extended to tackle variants of the CETSP, such as the Close-Enough Vehicle Routing Problem (CEVRP) [5] and the Generalized Close-Enough Traveling Salesman Problem (GCETSP) [39]. These variants introduce additional complexities and constraints, providing new challenges for algorithm design and optimization.

CRediT authorship contribution statement

Zhenyu Lei: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Writing – original draft. **Jin-Kao Hao:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Validation, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data are available online.

Acknowledgments

We are grateful to the reviewers for their useful comments that helped us to improve the work. We thank Dr. Andrea Di Placido of [3] for answering our questions and Prof. Béla Vizvári of [4] for providing the real-world instances on laser welding robot path planning. Support from China Scholarship Council (CSC, Grant No. 202206330014) for the first author is acknowledged.

References

[1] D.J. Gulczynski, J.W. Heath, C.C. Price, The close enough traveling salesman problem: A discussion of several heuristics, Perspectives in Operations Research: Papers in Honor of Saul Gass' 80 th Birthday, Springer, 2006, pp. 271–283.
[2] W. Mennell, B. Golden, E. Wasil, A steiner-zone heuristic for solving the close-enough traveling salesman problem, in: 2th INFORMS Computing Society Conference: Operations Research, Computing, and Homeland Defense, Monterey, California, January 9–11, 2011, pp. 1–23.
[3] A. Di Placido, C. Archetti, C. Cerrone, A genetic algorithm for the close-enough traveling salesman problem with application to solar panels diagnostic reconnaissance, Comput. Oper. Res. 145 (2022) 105831.
[4] A. Nedjati, B. Vizvari, Robot path planning by traveling salesman problem with circle neighborhood: Modeling, algorithm, and applications, Ind. Eng. 6 (2020) 288–293.
[5] W.K. Mennell, Heuristics for Solving Three Routing Problems: Close-Enough Traveling Salesman Problem, Close-Enough Vehicle Routing Problem, and Sequence-Dependent Team Orienteering Problem (Ph.D. thesis), University of Maryland, College Park, 2009.
[6] B. Behdani, J.C. Smith, An integer-programming-based approach to the close-enough traveling salesman problem, INFORMS J. Comput. 26 (3) (2014) 415–432.
[7] W.P. Coutinho, R.Q.d. Nascimento, A.A. Pessoa, A. Subramanian, A branch-and-bound algorithm for the close-enough traveling salesman problem, INFORMS J. Comput. 28 (4) (2016) 752–765.
[8] F. Carrabs, C. Cerrone, R. Cerulli, M. Gaudioso, A novel discretization scheme for the close enough traveling salesman problem, Comput. Oper. Res. 78 (2017) 163–171.
[9] F. Carrabs, C. Cerrone, R. Cerulli, C. D'Ambrosio, Improved upper and lower bounds for the close enough traveling salesman problem, in: 12th International Conference on Green, Pervasive, and Cloud Computing, Cetara, Italy, May 11-14, Springer International Publishing, Cham, 2017, pp. 165–177.
[10] Z. Yang, M.-Q. Xiao, Y.-W. Ge, D.-L. Feng, L. Zhang, H.-F. Song, X.-L. Tang, A double-loop hybrid algorithm for the traveling salesman problem with arbitrary neighbourhoods, European J. Oper. Res. 265 (1) (2018) 65–80.
[11] X. Wang, B. Golden, E. Wasil, A steiner zone variable neighborhood search heuristic for the close-enough traveling salesman problem, Comput. Oper. Res. 101 (2019) 200–219.
[12] F. Carrabs, C. Cerrone, R. Cerulli, B. Golden, An adaptive heuristic approach to compute upper and lower bounds for the close-enough traveling salesman problem, INFORMS J. Comput. 32 (4) (2020) 1030–1048.
[13] F. Alizadeh, D. Goldfarb, Second-order cone programming, Math. Program. 95 (1) (2003) 3–51.
[14] C. Cerrone, R. Cerulli, B. Golden, Carousel greedy: A generalized greedy algorithm with applications in optimization, Comput. Oper. Res. 85 (2017) 97–112.
[15] P. Moscato, C. Cotta, A modern introduction to memetic algorithms, in: Handbook of Metaheuristics, Springer, 2010, pp. 141–183.
[16] F. Neri, C. Cotta, Memetic algorithms and memetic computing optimization: A literature review, Swarm Evol. Comput. 2 (2012) 1–14.
[17] P. He, J.-K. Hao, Memetic search for the minmax multiple traveling salesman problem with single and multiple depots, European J. Oper. Res. 307 (3) (2023) 1055–1070.
[18] M. Kurdi, A memetic algorithm with novel semi-constructive evolution operators for permutation flowshop scheduling problem, Appl. Soft Comput. 94 (2020) 106458.
[19] J. Ren, J.-K. Hao, F. Wu, Z.-H. Fu, An effective hybrid search algorithm for the multiple traveling repairman problem with profits, European J. Oper. Res. 304 (2) (2023) 381–394.
[20] H.-B. Song, Y.-H. Yang, J. Lin, J.-X. Ye, An effective hyper heuristic-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem, Appl. Soft Comput. 135 (2023) 110022.
[21] J. Xiao, J. Du, Z. Cao, X. Zhang, Y. Niu, A diversity-enhanced memetic algorithm for solving electric vehicle routing problems with time windows and mixed backhauls, Appl. Soft Comput. 134 (2023) 110025.
[22] Y. Deng, Y. Liu, D. Zhou, An improved genetic algorithm with initial population strategy for symmetric TSP, Math. Probl. Eng. 2015 (2015) D 212794.
[23] J.-K. Hao, Memetic algorithms in discrete optimization, in: Handbook of Memetic Algorithms, Springer, 2012, pp. 73–94.
[24] Y. Nagata, S. Kobayashi, A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem, INFORMS J. Comput. 25 (2) (2013) 346–363.
[25] C.-K. Ting, Improving edge recombination through alternate inheritance and greedy manner, in: Evolutionary Computation in Combinatorial Optimization: 4th European Conference, EvoCOP 2004, Coimbra, Portugal, April 5-7, 2004. Proceedings 4, Springer, 2004, pp. 210–219.
[26] K. Sörensen, M. Sevaux, MA|PM: memetic algorithms with population management, Comput. Oper. Res. 33 (5) (2006) 1214–1225.
[27] D.C. Porumbel, J.-K. Hao, P. Kuntz, An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring, Comput. Oper. Res. 37 (10) (2010) 1822–1832.

[28] X. Gao, B. Xiao, D. Tao, X. Li, A survey of graph edit distance, Pattern Anal. Appl. 13 (2010) 113–129.
[29] P. Hansen, N. Mladenović, R. Todosijević, S. Hanafi, Variable neighborhood search: basics and variants, EURO J. Comput. Optim. 5 (3) (2017) 423–454.
[30] K. Helsgaun, An effective implementation of the Lin–Kernighan traveling salesman heuristic, European J. Oper. Res. 126 (1) (2000) 106–130.
[31] M. Baker, Alhazen's problem, Amer. J. Math. 4 (1) (1881) 327–331.
[32] G. Jarosch, J.S. Nimczik, I. Sorkin, Granular Search, Market Structure, and Wages, Tech. Rep., National Bureau of Economic Research, 2019.
[33] B. Bixby, G. Reinelt, TSPLIB, A traveling salesman problem library, ORSA J. Comput. 3 (1991) 376–384.
[34] I.-M. Chao, Algorithms and Solutions To Multi-Level Vehicle Routing Problems, University of Maryland, College Park, 1993.
[35] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual, 2023, URL https://www.gurobi.com.
[36] M. López-Ibáñez, J. Dubois-Lacoste, L.P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Oper. Res. Perspect. 3 (2016) 43–58.
[37] F. Wilcoxon, Individual Comparisons By Ranking Methods, Springer, 1992.
[38] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, European J. Oper. Res. 290 (2) (2021) 405–421.
[39] A. Di Placido, C. Archetti, C. Cerrone, B. Golden, The generalized close enough traveling salesman problem, European J. Oper. Res. 310 (3) (2023) 974–991.