ORIGINAL ARTICLE

Scalable parallel memetic algorithms for large close-enough traveling salesman problems

Deniz Canturk · Tansel Dökeroğlu

Received: 27 March 2025 / Accepted: 6 December 2025
© The Author(s) 2026

Abstract
The Close-Enough Traveling Salesman Problem (CETSP) is a complex variant of the classical TSP in which each target is represented by a disk-shaped neighborhood, and a visit is considered valid if any point within that neighborhood is reached. This formulation provides a more realistic model for real-world applications such as surveillance, wireless sensor routing, and Unmanned Aerial Vehicle (UAV) path planning. However, due to its NP-hard nature, solving large-scale CETSP instances efficiently remains a significant challenge. The main objective of this study is to develop scalable and robust Parallel Memetic Algorithms (PMAs) that enhance computational efficiency and solution quality for large CETSP instances. Two complementary strategies are proposed: (i) a Multi-Population PMA that maintains diversity through distributed crossover and parallel local optimization, and (ii) a Single-Population, Multi-Fitness PMA that accelerates convergence via simultaneous parallel fitness evaluations. Extensive experiments conducted on 42 benchmark instances demonstrate that the proposed algorithms outperform state-of-the-art approaches, achieving 21 new best-known solutions and up to a 10–15 times reduction in the execution times compared to the sequential MA-CETSP algorithm. The results confirm that the Multi-Population PMA provides superior exploration capabilities for complex instances, while the Single-Population PMA delivers faster convergence with comparable solution quality. Overall, this study establishes new performance benchmarks for the CETSP and demonstrates the effectiveness of parallel memetic algorithms in solving large-scale combinatorial optimization problems.

Keywords Close-enough traveling salesman problem · Optimization · Parallel computation · Scalability

1 Introduction

The Close-Enough Traveling Salesman Problem (CETSP) is a variant of the well-known TSP [1–3]. Unlike the standard TSP, where a salesman must visit a set of target locations exactly, in the CETSP, each target location is associated with a predefined neighborhood. The objective is to determine the shortest Hamiltonian cycle that starts and ends at a depot while passing through at least one point in each target's neighborhood.

Let $V = \{v_1, v_2, \ldots, v_N\}$ be a set of N target points in a Euclidean plane, and let p_0 denote the depot. Each target v_i has an associated disk-shaped neighborhood $\mathcal{N}_i$ with radius r_i. The goal is to minimize the total travel

Published online: 30 January 2026

distance while ensuring each target is covered by visiting a point within its neighborhood. The CETSP is formally defined as follows:

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

$$\text{subject to} \quad S = \{p_0, p_1, \ldots, p_N, p_0\}, \quad p_i \in \mathcal{N}_i, \quad i = 1, \ldots, N. \tag{2}$$

where $d(x, y)$ represents the Euclidean distance between points x and y.

Figure 1 represents the tours of CETSP (black lines) and TSP tours (green lines), including targets A, B, and C. The black triangle is a depot, the red nodes are the targets with their disk neighborhood. As can be seen, the CETSP tour is shorter than the TSP tour.

The CETSP has numerous practical applications, including: Automated meter reading using RFID technology [4], Solar panel diagnostic reconnaissance [5], Robot path planning and navigation [6], and Unmanned Aerial Vehicle (UAV) mission planning [7] for reconnaissance and delivery. The CETSP extends the classical TSP by allowing flexible target visitation within predefined neighborhoods. Solving the CETSP is challenging due to its NP-hard nature. Its computational complexity necessitates advanced heuristic and optimization techniques to achieve efficient solutions for large and real-world applications. Various metaheuristic and exact methods have been developed, including: Population-based genetic algorithms, Variable Neighborhood Descent (VND) optimization, Mixed-Integer Linear Programming (MILP) approaches.

The motivation for this study arises from the computational challenges of solving large-scale instances of the CETSP, which remains NP-hard and highly demanding in terms of time and resources. Existing heuristic and exact approaches often fail to provide efficient solutions within reasonable execution times. Therefore, this research aims to develop scalable and robust parallel memetic algorithms that exploit multi-core architectures to enhance computational efficiency, improve solution quality, and effectively address the limitations of sequential optimization methods.

Parallel Memetic Algorithms (PMAs) exhibit greater robustness compared to their sequential counterparts due to enhanced population diversity, improved exploration-exploitation balance, and reduced susceptibility to premature convergence. By distributing computational tasks across multiple processors, PMAs maintain diverse solutions, mitigating stagnation in local optima [8–11]. Additionally, parallelism enables more effective crossover and mutation strategies, fostering a more thorough search of the solution space. The increased computational efficiency of PMAs allows for the execution of more extensive local searches within feasible time constraints, improving solution quality. Empirical studies demonstrate that PMAs consistently outperform sequential variants

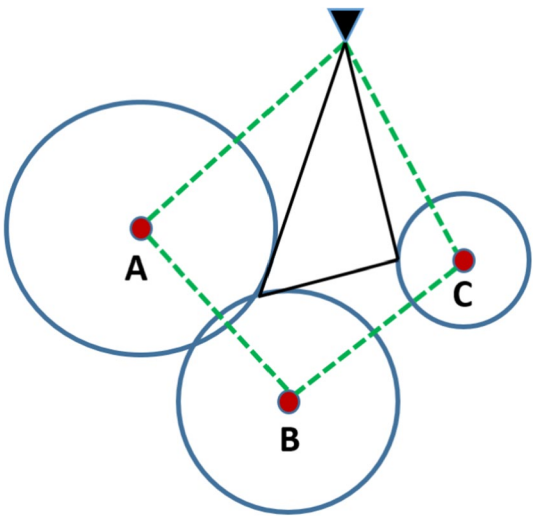


Fig. 1 CETSP tour (black lines) and TSP tour (green lines) of a sample TSP for locations, A, B, and C.

in complex optimization problems, highlighting their superiority in robustness and adaptability. We proposed two new parallel versions of the memetic algorithm designed by Lei and Hao [12]. The algorithm proposed by Lei & Hao is one of the best algorithms in the literature, reporting new best solutions with reasonable execution times. We further improve the performance of this algorithm using OpenMP libraries [13–15].

1.1 Our contributions

We present the first parallel memetic algorithms (PMAs) specifically tailored for solving the CETSP, leveraging modern multi-core processors to significantly improve computational efficiency. To this end, we introduce two complementary parallelization strategies. The first is a multi-population PMA that employs distributed crossover, mutation, and local improvement procedures to maintain high population diversity and effectively mitigate premature convergence. The second is a single-population, multi-fitness evaluation PMA that accelerates convergence by evaluating multiple fitness measures in parallel. Both strategies are reinforced by robust local search components that provide a strong balance between exploration of the solution space and exploitation of high-quality candidate solutions.

A central focus of our work is the effective handling of large-scale CETSP instances, where the exponential growth of the search space poses substantial challenges to traditional approaches. The proposed algorithms are particularly well-suited for these demanding scenarios, consistently delivering superior performance. Extensive experiments conducted on 42 benchmark instances demonstrate that our methods outperform state-of-the-art techniques and achieve 21 new best-known solutions. In the case of the largest problem instances, our results establish new reference values for the CETSP, setting a strong benchmark for future research in this domain.

Related studies are reported in Sect. 2. Our proposed algorithms are introduced in Sect. 3. The details of the performance evaluation of the experiments and the comparisons of new parallel algorithms to state-of-the-art algorithms are presented in Sect. 4. The last section provides concluding remarks and future works.

2 Related work

This section provides a concise yet comprehensive summary of the best algorithms proposed in the literature for the CETSP. To the best of our knowledge, we noticed a lack of parallel algorithms like ours, so we included them as well.

Gulczynski et al. explored the CETSP in applications like radio frequency identification (RFID) meter reading, where a reader only needs to get within a certain radius of each customer in 2006 [16]. The study presents various heuristics developed by graduate students to find near-optimal solutions. These heuristics are surveyed, and their effectiveness is evaluated on a diverse set of test cases. Carrabs et al. introduced a new discretization scheme to compute both lower and upper bounds for the optimal solution [17]. Additionally, a graph reduction algorithm is applied to reduce problem size and improve computational efficiency. The approach is tested on benchmark instances, showing that it outperforms existing methods in speed and provides more accurate bounds in most cases.

Deckerova et al. introduced a novel improvement heuristic for the CETSP with obstacles (CETSPobs), a TSP variant where visits are made to disk-shaped regions while ensuring a collision-free path in a polygonal domain [18]. A new Post-Optimization approach using Mixed Integer Non-Linear Programming (MINLP) enhances heuristic solutions. Evaluations show significant improvements across solvers, with sparse sampling yielding the best results. The findings highlight the effectiveness of the MINLP-based approach for continuous CETSPobs optimization. Placido et al. proposed a Genetic Algorithm (GA) [19], incorporating 2-opt, second-order cone programming, and a bisection algorithm for optimization [5]. Tested on 62 benchmark instances, GA outperforms

or matches state-of-the-art methods and finds 32 new best solutions. Additionally, new metrics are introduced to assess problem complexity, refining the Overlap Ratio (OR) metric. A case study on solar panel diagnostics demonstrates the effectiveness of the GA-based approach.

Coutinho et al. introduced an exact algorithm that combines Branch-and-Bound (B&B) with second-order cone programming [20]. The algorithm is evaluated on many benchmark instances from the literature, successfully obtaining optimal solutions for previously unsolved cases with up to 1,000 vertices. The study considers instances in both two- and three-dimensional spaces. Behdani & Smith studied the CETSP and proposed a MINLP model using a discretization approach, providing both lower and upper bounds on the optimal tour length [21]. To improve solvability, the authors developed valid inequalities and introduced two alternative formulations: one enhancing the lower bound and another restructuring the problem into a more efficient two-stage framework. Computational results validate the effectiveness of our approach.

Carrabs et al. employed an effective discretization strategy and the Carousel Greedy algorithm to iteratively select neighborhoods, constructing a feasible solution step by step [22]. Their heuristic efficiently computes tight upper and lower bounds on the optimal solution. Computational experiments on benchmark instances demonstrate that the new method frequently finds the optimal solution and, in general, provides more accurate upper bounds than existing algorithms.

Placido et al. introduced the Generalized CETSP in 2023, where each customer is surrounded by multiple disks of varying radii, with higher rewards for closer visits [3]. A new GA is proposed to solve the problem efficiently, and its performance is evaluated on benchmark CETSP and TSP instances. Results demonstrate that the GA produces high-quality solutions with short computation times, outperforming alternative approaches. The study also compares the GA's solutions with those obtained using a method that pre-selects intersection points on customer disks. Findings indicate that the GA effectively balances prize collection and route length, making it a promising approach for GCETSP. Deckerova et al. investigated the Generalized Traveling Salesman Problem with Neighborhoods (GTSPN), where the objective is to find the shortest path visiting given neighborhood sets represented as convex regions [23]. A novel B&B approach combined with Mixed-Integer Second-Order Cone Programming (MISOCP) is proposed to determine the first lower bounds and solve the GTSPN optimally. The results demonstrate that the method provides tight lower bounds and can solve instances with up to 15 neighborhoods within a limited computational time, making it suitable for high-dimensional inspection tasks with robotic manipulators.

Yang et al. proposed a boundary-based encoding scheme and a hybrid algorithm combining Particle Swarm Optimization (PSO) [24, 25] for continuous position search and a GA for optimizing the visiting sequence [26]. This approach significantly reduces the search space without compromising solution quality. Computational results demonstrate its efficiency in finding high-quality solutions, outperforming existing algorithms and solving real-world applications. Lei & Hao presented a memetic algorithm combining a specialized crossover operator with an effective local optimization procedure [12]. Tested on 62 benchmark instances, the algorithm proves highly competitive, achieving 30 new best upper bounds. Its effectiveness is further demonstrated in a laser welding robot path planning problem. Additionally, insights into the algorithm's design are provided. Cariou et al. studied a sub-optimal trajectory for an Unmanned Aerial Vehicle collecting data from buried sensor nodes, an evolutionary algorithm combined with geometrical heuristics is proposed [27]. The effectiveness of the approach is demonstrated through scenarios with 15 and 50 targets, analyzing the total route length and discussing future research directions.

Faigl et al. introduced the Growing Self-Organizing Array (GSOA), a novel unsupervised learning procedure inspired by self-organizing maps for solving the TSP with Neighborhoods (TSPN) [28]. The method efficiently integrates combinatorial and continuous optimization, outperforming previous self-organizing approaches in both solution quality and computational time. Applied to the CETSP, GSOA delivers competitive solutions while being significantly faster than existing heuristics, making it a promising approach for routing problems with flexible waypoint selection. Carrabs et al. proposed a Mixed Integer Linear Programming (MILP) formulation in 2025

[29]. In his thesis, Toman proposed a system that replaces the SOCP solver in the B&B method with the Iteratively Refined Informed Sampling (IRIS) algorithm to handle the continuous part of the CETSP. This generalized IRIS approach, combined with region division and a fast sequence rejection technique, enables efficient solving of both standard and obstacle-constrained CETSP instances with guaranteed solution quality. [30]. Dokeroglu & Canturk proposed a new Harris Hawk Optimisation (HHO) metaheuristic for the solution of the CETSP [31].

Table 1 summarizes the list of algorithms in the literature that have been proposed for the solution of the CETSP. In this study, we made performance comparisons with algorithms that use the same problem instances as us. To the best of our knowledge, there exist no parallel algorithms to solve the CETSP that are comparable to our proposed algorithms.

3 Proposed parallel memetic algorithms

In this section, we provide a detailed description of the proposed *Parallel Memetic Algorithms (PMAs)*. These algorithms extend the sequential memetic algorithm developed by Lei and Hao [12], which is recognized as one of the most effective memetic approaches in the literature. To enhance scalability and computational performance, we introduce two novel PMA variants that leverage the capabilities of the OpenMP parallel programming framework [34].

The **first PMA** adopts a *multi-population strategy*. Rather than accelerating a single evolutionary process, this algorithm distributes multiple independent populations across parallel threads (hereafter referred to as worker nodes). This design improves diversification by allowing each subpopulation to explore different regions of the search space concurrently, thereby reducing the risk of premature convergence and improving global exploration.

The **second PMA** employs a *single-population framework*, in which a master thread centrally manages the main population while computationally intensive fitness evaluations are offloaded to worker threads. This master–worker architecture provides efficient resource utilization and accelerates the evolutionary process without compromising solution quality.

Both PMAs were implemented using the **OpenMP** API, which extends C and C++ to support shared-memory parallelism through compiler directives and runtime routines [13, 14]. OpenMP enables the development of efficient multithreaded applications that exploit the full computational power of modern multicore processors. Its scalable programming model dynamically spawns worker threads from a master thread to execute concurrent tasks. This architecture makes OpenMP particularly suitable for parallel metaheuristic algorithms, such as memetic algorithms, which integrate global and local search strategies for robust optimization.

Table 1 Proposed algorithms in the literature for the solution of the CETSP

Algorithm	Year	Method	Characteristics
Gulczynski et al. [16]	2006	Heuristics	3-phase heuristics, Steiner-zone
Behdani & Smith [21]	2014	Exact algorithm	MIP, discretization
Exact B&B [20]	2016	Exact algorithm	B&B, SOCP
Carrabs et al. [17]	2017	Heuristics	discretization, MIP
HA [26]	2018	Heuristics	PSO, GA
SZVNS [32]	2019	Heuristics	VNS, Steiner-zone
(ul/bl)Alg [22]	2020	Heuristics	discretization, carousel greedy algorithm
GA [5]	2022	Heuristics	GA, SOCP
Exact B&B [23]	2023	Heuristics	B&B, MILP Second-Order Cone
Mennell et al. [33]	2024	Heuristics	3-phase heuristics, Steiner-zone, SOCP
Memetic [12]	2024	Heuristics	crossover, VND
Harris Hawk [31]	2025	Metaheuristic	crossover, VND
MILP [29]	2025	Linear programming	discretization, graph reduction

The **tasking model** in OpenMP provides fine-grained workload distribution and dynamic scheduling, which is advantageous for metaheuristic approaches requiring adaptive computation, such as dynamically resizing populations or refining elite solutions. By leveraging these features, the proposed PMAs achieve strong scalability across diverse hardware architectures, resulting in accelerated optimization and improved solution quality.

Within the proposed PMAs, the **crossover** and **mutation operators** use distinct random seeds across processors to encourage exploration in diverse regions of the solution space. This approach increases population diversity and mitigates premature convergence. Furthermore, the **local search mechanism**–executed concurrently across processors–enables comprehensive refinement of multiple candidate solutions simultaneously. By combining global exploration (via crossover and mutation) with local exploitation (via adaptive refinement), the PMAs achieve faster convergence and higher-quality solutions.

Overall, this parallelized framework enhances both the robustness and computational efficiency of the memetic algorithm, making it well-suited for solving complex combinatorial optimization problems such as the CETSP.

3.1 Main framework of our proposed memetic algorithms

In this subsection, we present the underlying structure of our sequential Memetic Algorithm (MA), which serves as the foundation for the proposed parallel versions. This algorithm is an extended implementation of the MA-CETSP proposed by Lei and Hao [12], widely recognized for its strong balance between exploration and exploitation. The overall workflow of the algorithm is described in Algorithm 1, and its major components are detailed below.

The algorithm begins by **preprocessing the input instance**, where dominated targets–those that can be covered by the depot or by other targets–are removed. This reduction step improves computational efficiency and solution quality [5, 32]. Next, the **population initialization** stage employs **K-means clustering** to group targets into clusters, generating an initial set of feasible tours. Within each cluster, the exact visiting positions are selected randomly in polar coordinates to reduce the occurrence of long edges and ensure geometric diversity. The best individual among the initial solutions is then recorded as the current best solution S^*.

After initialization, the algorithm proceeds into an **iterative evolutionary loop**, where new offspring are generated, evaluated, and integrated into the population until the termination criteria are satisfied.

At each iteration:

- Two parent solutions are randomly selected from the population.
- The **Multi-Step Crossover (MSX)** operator recombines these parents using **AB-cycles**, a mechanism inspired by the Generalized Alternating Crossover (GAX) and Edge Assembly Crossover (EAX) crossovers. MSX iteratively exchanges m consecutive edges between the parents, ensuring that connectivity and feasibility are preserved. This operator promotes population diversity while maintaining computational complexity $O(n)$.
- The offspring is then subjected to a **Swap mutation** operator, which exchanges two nodes in the visiting sequence. This mutation is applied with a predefined probability that depends on the stagnation count and a patience parameter, preventing premature convergence.
- A **Variable Neighborhood Descent (VND)**-based local search is applied to the mutated solution to perform fine-grained refinement and enhance local exploitation.
- **Population management** then replaces inferior individuals and ensures diversity using the **graph edit distance** metric to measure dissimilarity between solutions. A distance threshold is enforced to avoid duplicates and maintain a healthy balance between intensification and diversification.

If the new best individual S_{best} improves upon S^*, it replaces the current best; otherwise, the stagnation counter increases. The process continues until reaching either the maximum number of generations ($Iter_{max}$) or the

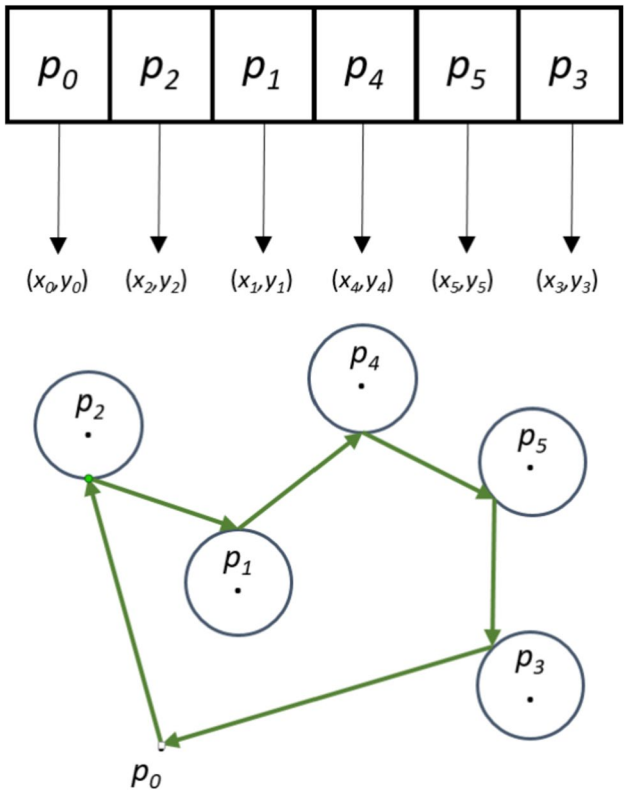


Fig. 2 The chromosome structure of our proposed algorithm.

maximum number of consecutive stagnation iterations (ρ). When the termination condition is met, the algorithm returns the best-found solution S^*.

The chromosome structure, illustrated in Fig. 2, encodes a solution as a sequence of locations P_i with coordinates (x_i, y_i). The order of these genes determines the sequence of visiting the nodes, and evolutionary operators iteratively refine order (with gene swaps) to minimize the total travel cost while satisfying proximity constraints. The chromosome $p_0, p_2, p_1, p_4, p_5, p_3$ represents the order of the locations as given in Fig. 2.

3.2 Local optimization, Variable Neighborhood Descent (VND)

Our proposed algorithms integrate local optimization through VND [35], executed concurrently across multiple threads to refine solutions by systematically exploring the search space. The procedure alternates between two complementary optimization phases: **sequence-only optimization** using the Lin–Kernighan Heuristic (LKH solver) [36] and **position-only optimization** via Disk-Constrained Position Optimization (DCPO) [37]. This alternating mechanism maintains an effective balance between improving visiting sequences and fine-tuning positional accuracy. DCPO is applied twice–before and after the LKH solver–to further enhance positional adjustments. Subsequently, **joint optimization methods** (Joint-Relocate and Joint-Swap) refine both sequence and position simultaneously. Finally, **Sequence-Only Constrained Position Optimization (SOCP)** [38] is applied to fine-tune the solution and achieve maximum accuracy. This structured and hierarchical approach ensures intensive local exploitation, resulting in more precise and high-quality solutions.

Algorithm 2 presents the VND-based local search framework used to solve the continuous-space optimization problem of determining exact visiting positions. The algorithm follows a structured sequence of improvement phases designed to progressively enhance the current solution. It begins with position optimization (DCPO) while keeping the sequence fixed, followed by sequence optimization (LKH) with fixed positions. The DCPO step is

executed twice: first to refine the visiting positions of the offspring solution generated by crossover–serving as a preprocessing step for the LKH solver–and second to fine-tune positions based on the sequence refined by LKH, preparing for subsequent joint optimization. Next, the algorithm employs **Joint-Relocate** and **Joint-Swap** operators to jointly adjust both sequence and positions, improving overall route structure and spatial consistency. Finally, SOCP is applied to the optimized sequence to yield the final refined solution. This systematic sequence of alternating and joint refinements ensures comprehensive exploration of the solution space and results in a more accurate and robust outcome.

Algorithm 1 Framework of the Sequential Memetic Algorithm (MA-CETSP)

Require: Instance I, maximum number of generations $Iter_{max}$, patience for stagnation generations ρ.
Ensure: Best-found solution S^*.
$Iter \leftarrow 0$, $Iter_{st} \leftarrow 0$
$I \leftarrow Preprocessing(I)$ ▷ Remove dominated targets and simplify instance
$P \leftarrow Initialization(I)$ ▷ Generate initial population using K-means clustering
$S^* \leftarrow BestSolution(P)$
while $Iter < Iter_{max}$ **and** $Iter_{st} < \rho$ **do**
  $S_A, S_B \leftarrow RandomParents(P)$
  $S' \leftarrow MSX(S_A, S_B)$ ▷ Apply Multi-Step Crossover
  $S' \leftarrow Mutation(S')$ ▷ Swap two nodes to diversify search
  $S' \leftarrow VND(S')$ ▷ Refine solution using local search
  $P \leftarrow PopulationManagement(S', P)$
  $S_{best} \leftarrow BestSolution(P)$
  if $f(S_{best}) < f(S^*)$ **then**
    $S^* \leftarrow S_{best}$
    $Iter_{st} \leftarrow 0$
  else
    $Iter_{st} \leftarrow Iter_{st} + 1$
  end if
  $Iter \leftarrow Iter + 1$
end while
return S^*

Algorithm 2 Variable Neighborhood Descent-Based Local Search

Require: Initial solution S
Ensure: Improved solution S'
$S' \leftarrow DCPO(S)$
$S' \leftarrow LKH(S')$
$S' \leftarrow DCPO(S')$
$S' \leftarrow JointRelocate(S')$
$S' \leftarrow JointSwap(S')$
$S' \leftarrow SOCP(S')$
return S'

3.3 Multi-population parallel memetic algorithm

The Multi-Population Parallel Memetic Algorithm (Multi-Population-PMA) is a robust optimization approach that leverages parallel computing to enhance the search process for solving complex optimization problems [39]. The fundamental concept of Multi-Population-PMA is to maintain multiple independent populations, each evolving separately on a dedicated thread. This independence enables each population to explore distinct regions of the solution space concurrently, thereby promoting diversity and reducing the likelihood of premature convergence.

Each thread executes its own instance of the memetic algorithm, with no direct communication between populations during the search. This lack of inter-population exchange ensures that each population evolves autonomously, maintaining high diversification throughout the optimization process. Diversity is a key factor in global optimization, as it prevents the algorithm from being trapped in local optima. By exploring different regions of the search space simultaneously, the algorithm increases its potential to identify near-optimal or globally optimal solutions.

This parallel multi-population structure is particularly advantageous for large-scale or multimodal problems, where the search landscape is complex and computationally demanding. By distributing the search across multiple threads, the Multi-Population-PMA achieves improved coverage of the solution space, scalability, and computational efficiency. Algorithm 3 presents the detailed framework of the proposed Multi-Population-PMA implemented using OpenMP for solving large instances of the CETSP.

Algorithm 3 extends the sequential memetic algorithm, where multiple independent populations evolve simultaneously instead of sharing a single global population. After initialization (Lines 4–7), the algorithm creates N separate populations–one per thread–allowing each to explore a different region of the search space (Line 8). The key modification appears in Lines 9–17, where the entire evolutionary process (selection, crossover, mutation, and VNDS-based local search) is executed in parallel for each population using OpenMP. Unlike the single-population PMA, no fitness-sharing or migration occurs between populations during evolution, which preserves diversification and reduces synchronization overhead.

Once all threads complete their local search phase, a global synchronization barrier is applied (Line 18), after which the best solution among all populations is evaluated (Line 19). The global stagnation logic (Lines 20–26) is handled only at the master thread, meaning the stopping condition depends on the collective progress rather than individual populations. The algorithm continues until the iteration or stagnation limits are reached, and finally returns the best solution found across all populations (Line 27).

Algorithm 3 Main Framework of the Multi-Population-PMA with OpenMP

Require: Instance I,
Maximum number of generations $Iter_{max}$,
Number of populations N (equal to the number of threads),
Patience for stagnation generations ρ.
Ensure: The best solution S^*.
$Iter \leftarrow 0$, $Iter_{st} \leftarrow 0$
$I \leftarrow Preprocessing(I)$
$P_1, P_2, \ldots, P_N \leftarrow InitializePopulations(I, N)$
$S^* \leftarrow BestSolution(P_1, P_2, \ldots, P_N)$
#pragma omp parallel for num_threads(N) ▷ Parallel evolution of independent populations
for each population P_i **do**
    while $Iter \leq Iter_{max}$ and $Iter_{st} \leq \rho$ **do**
        $S_A, S_B \leftarrow RandomParents(P_i)$
        $S' \leftarrow MSX(S_A, S_B)$ ▷ Multi-Step Crossover
        $S' \leftarrow Mutation(S')$ ▷ Apply mutation
        $S' \leftarrow VNDS(S')$ ▷ Local search with Variable Neighborhood Descent
        $P_i \leftarrow PopulationManagement(S', P_i)$ ▷ Update population P_i
    end while
end for
#pragma omp barrier ▷ Synchronize threads before global evaluation
$S_{best} \leftarrow BestSolution(P_1, P_2, \ldots, P_N)$
if $f(S_{best}) < f(S^*)$ **then**
    $S^* \leftarrow S_{best}$
    $Iter_{st} \leftarrow 0$ ▷ Reset stagnation counter
else
    $Iter_{st} \leftarrow Iter_{st} + 1$ ▷ Increment stagnation counter
end if
$Iter \leftarrow Iter + 1$
return S^*

3.4 Single-population multiple fitness parallel memetic algorithm (Single-Population-PMA)

We developed a single-population, multi-threaded fitness Parallel Memetic Algorithm (Single-Population-PMA) that exploits shared-memory parallelism to enhance both computational speed and solution quality [40, 41]. In this framework, a master thread maintains a single global population, while the fitness evaluation of newly generated individuals is delegated to multiple worker threads via OpenMP. After candidate solutions are produced, they are broadcast to the worker threads, which compute fitness values in parallel. Once these evaluations are completed, the master thread collects the results and integrates only the most promising individuals into the population. This design enables extremely fast exploitation, allowing the algorithm to evaluate a significantly larger number of solutions within the same time budget compared to its sequential counterpart.

A major strength of the Single-Population-PMA is its ability to reach generation counts that are unattainable in conventional evolutionary algorithms, where fitness evaluation is a computational bottleneck. By parallelizing the most expensive stage of the evolutionary cycle, the algorithm achieves faster convergence and improved optimization accuracy. The method preserves the key advantages of memetic algorithms–namely, the combination of global evolutionary search and local refinement–while leveraging parallel execution to scale efficiently on modern multi-core hardware. As a result, the proposed approach consistently outperforms traditional evolutionary

algorithms in both runtime and solution quality. The complete procedure is summarized in Algorithm 4, which presents the Single-Population-PMA used for solving large-scale CETSP instances.

Algorithm 4 introduces a second form of parallelism by employing parallel fitness evaluation while keeping all other components identical to the original framework. The evolutionary loop (Lines 5–23) follows the same sequence of operations–parent selection, MSX crossover, mutation, and local search–but the main modification occurs in Lines 9–13, where the fitness of newly generated individuals is evaluated concurrently using OpenMP (#pragma omp parallel for). This allows the most computationally expensive stage of the algorithm to be distributed across (T) threads, reducing runtime without altering solution quality. Unlike the multi-population version, the algorithm maintains a single global population and does not require migration or inter-population synchronization. After parallel evaluation, the VND local search (Line 14) and population management (Line 15) proceed as in the sequential model. Thus, the single-population PMA provides a lightweight parallelization strategy that accelerates execution while preserving the original algorithmic behavior.

Algorithm 4 Single population Parallel Memetic Algorithm Framework (Single-Population-PMA)

Require: Instance I, Maximum number of generations $Iter_{max}$, Patience for stagnation generations ρ, Number of threads T.
Ensure: The best solution S^*.

1: $Iter \leftarrow 0$, $Iter_{st} \leftarrow 0$
2: $I \leftarrow Preprocessing(I)$
3: $P \leftarrow Initialization(I)$
4: $S^* \leftarrow BestSolution(P)$
5: **while** $Iter \leq Iter_{max}$ and $Iter_{st} \leq \rho$ **do**
6: $S_A, S_B \leftarrow RandomParents(P)$
7: $S' \leftarrow MSX(S_A, S_B)$
8: $S' \leftarrow Mutation(S')$
9: **Parallel Fitness Calculation:**
10: **#pragma omp parallel for num_threads**(T)
11: **for** each individual S_i in S' **do**
12: $f(S_i) \leftarrow FitnessCalculation(S_i)$ ▷ Fitness calculated in parallel
13: **end for**
14: $S' \leftarrow VNDS(S')$ ▷ Local search applied after fitness evaluation
15: $P \leftarrow PopulationManagement(S', P)$
16: $S_{best} \leftarrow BestSolution(P)$
17: **if** $f(S_{best}) < f(S^*)$ **then**
18: $S^* \leftarrow S_{best}$
19: **else**
20: $Iter_{st} \leftarrow Iter_{st} + 1$
21: **end if**
22: $Iter \leftarrow Iter + 1$
23: **end while**
24: **return** S^*

3.5 Local optima and stagnation prevention using parallelism

Local optima refer to solutions in an optimization problem that is better than their neighboring solutions but are not the best possible solutions globally [42]. In other words, a local optimum is a maximum or a minimum within a limited region of the search space, but there may exist other, higher peaks or lower valleys elsewhere that represent the global optimum. Getting trapped in a local optimum is a common challenge, as algorithms may converge

too early to a suboptimal solution and fail to explore better regions [43]. This issue is particularly problematic in complex, multimodal search spaces with multiple local optima. To overcome this, techniques such as mutation, crossover, adaptive parameter tuning, and hybridization with local search methods are used to escape local optima and continue searching for better solutions. Stagnation to local optima in optimization refers to a state where the algorithm stops making significant progress over successive generations and stuck into local optima. Several factors contribute to stagnation. One major cause is over-exploitation, where the algorithm focuses too much on refining high-performing solutions but lacks exploration of new regions in the search space. Strong selection pressure can also lead to stagnation, as it favors only the best individuals while eliminating diversity too quickly. Additionally, low mutation rates and poorly tuned parameters can slow down the introduction of new genetic variations, further reducing the algorithm's ability to escape stagnation.

In the PMAs we have proposed, stagnation prevention plays a crucial role in maintaining diversity and improving convergence speed across multiple processing units. Since these algorithms distribute computations across different populations, preventing stagnation ensures that these subpopulations do not become too similar, thereby enhancing exploration and avoiding premature convergence. One key advantage of stagnation prevention is that it helps maintain diversity across parallel populations. Techniques such as adaptive update strategies of populations allow individuals to move between populations at optimal intervals, preventing isolated groups from stagnating in local optima. Additionally, asynchronous evaluation of individuals ensures that slower computations do not delay population updates, keeping the evolutionary process dynamic.

3.6 Computational complexity and execution time analysis of the proposed algorithms.

The computational complexity of the proposed PMAs can be expressed in terms of the number of targets N, the number of generations G, and the number of threads T. The main evolutionary cycle involves selection, crossover, mutation, fitness evaluation, and local optimization using the VND procedure. The sequential version (MA-CETSP) operates with a total complexity of $\mathcal{O}(G \times N^2)$ due to pairwise distance computations and local refinements. In the parallel versions, these operations are distributed across T threads. Thus, the theoretical complexity reduces to $\mathcal{O}\left(\frac{G \times N^2}{T}\right)$, assuming ideal load balancing. Experimental results confirm near-linear speed-up in both the Multi-Population and Single-Population PMAs, with efficiency between 85% and 95%. The Multi-Pop-PMA exhibits higher computational cost $\mathcal{O}(T \times G \times N^2)$ due to multiple populations but achieves superior exploration, while the Single-Pop-PMA attains faster convergence. Execution time analysis (Table 4 in the paper) shows that the Single-Pop-PMA achieves the lowest runtime–up to 10–15 times faster than the sequential MA-CETSP–while maintaining comparable or better solution quality, demonstrating the scalability and computational efficiency of the proposed parallel framework.

4 Experimental setup and evaluation of the results

We implemented our algorithms in OpenMP/C++, and the experiments were conducted on a workstation equipped with an Intel(R) Xeon(R) Silver 4210R processor running at 2.40 GHz with 20 cores and 64 GB of RAM, operating on Microsoft Windows 11 Pro for Workstations. To solve the second-order cone programming (SOCP) component of the algorithm, we utilized Gurobi 10 [44]. All experiments were performed using 16 threads.

Our algorithm (an enhanced version of MA-CETSP) utilizes five key parameters: population size N_p, maximum generations $Iter_{\max}$, fitness coefficient λ, distance threshold μ, and nearest neighbors size κ. To optimize these parameters, we employed the IRACE tool [45], selecting nine diverse instances for training with a tuning budget of 1000. Table 2 provides details of the parameters, their data types, and the optimal configurations identified by IRACE. These optimal values were used in all our experiments. Additionally, we established a relationship between the stagnation patience ρ and $Iter_{\max}$, setting $\rho = Iter_{\max}/10$ based on experimental analysis. This

Table 2 Optimized parameter configuration used in our proposed algorithm

Parameter	Type	Optimal value
N_p	Categorical	20
$Iter_{\max}$	Categorical	5000
λ	Real	0.96
μ	Integer	5
κ	Categorical	50
Thread count	Thread	16

ensures that our stopping criterion aligns with the GA reference algorithm [5]. Each instance was executed across ten independent runs.

4.1 Problem instances used in the experiments

The proposed algorithms are evaluated on 42 popular benchmark instances with sizes ranging from 10 to 1000 targets [33]. These instances are obtained from three sources: TSBLIB [46], Teams, and Geometric. 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 include: Two graphs with uniformly generated nodes (d493, dsj1000), Two graphs with clustered nodes (kroD100, lin318), Two drill-press problems (pcb442, rat195), and One graph where nodes are arranged in rough lines (rd400). The Teams subset consists of 14 instances from 7 graphs whose names contain “team” or "bonus," as discussed in [16]. The subset includes three graphs with uniformly generated points and four graphs with clustered points. Geometric subset consists of 20 instances from [33], including: 9 Bubbles instances where nodes are arranged in overlapping concentric squares, 5 Concentric-Circles instances where nodes are arranged in concentric circles, 5 RotatingDiamonds instances where nodes are arranged in non-overlapping concentric squares, one chaoSingleDep instance based on the problem given in [47].

The overlap ratio (OR) is a metric introduced to reflect CETSP instances’ characteristics and estimate the potential improvement over the TSP solution [5, 33]. 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:

$$OR = \frac{\sum_{i=1}^{N} r_i}{N \times L}. \tag{3}$$

Figures 3, 4, and 5 show the visualization of our solutions for the problem instances onus1000rdmRad, rd400rdmRad, and team4_400rdmRad respectively.

Table 3 presents the comparison of algorithms with sequential MA-CETSP. Bold results are the new best solutions reported by our PMAs. The performances of the MA-CETSP (our execution results), Multi-Pop-PMA, and Single-Pop-PMA algorithms are compared across 42 problem instances. This comparison highlights the differences between each algorithm’s results and the Best-Known Solutions (BKS), as well as their relative performances. Both Multi-Pop-PMA and Single-Pop-PMA algorithms have succeeded in finding new best solutions. The Multi-Pop-PMA algorithm outperformed the BKS in 19 instances. Single-Pop-PMA reported 2 new best solutions. These results indicate that Multi-Pop-PMA is more effective across a wider range of problems. In terms of overall performance, the Multi-Pop-PMA algorithm consistently outperformed both Single-Pop-PMA and MA-CETSP. This suggests that the multi-population structure of Multi-Pop-PMA allows for better exploration of the solution space, leading to superior results.

4.2 Execution time analysis of the proposed parallel memetic algorithms

In Multi-Pop-PMA, fitness calculations of multiple populations are performed by 16 threads simultaneously, leading to significant execution time improvements. Since each thread processes a different population independently, the workload is well-balanced in most cases. However, certain populations converge prematurely to

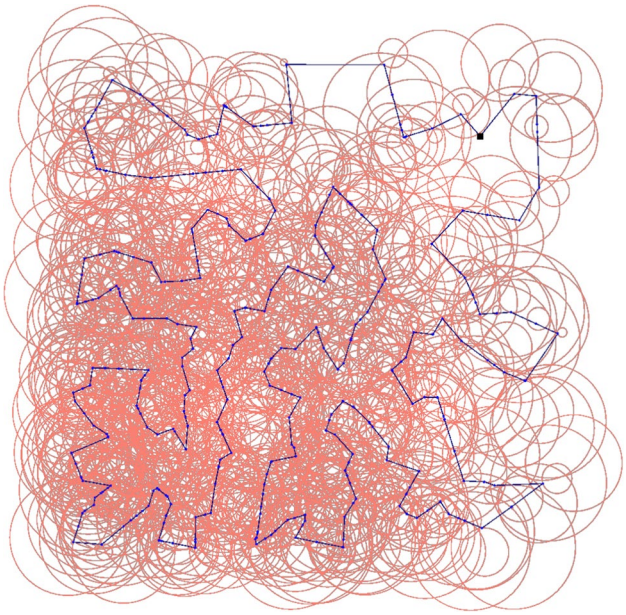


Fig. 3 Visualization of our solution for the bonus1000rdmRad problem instance.

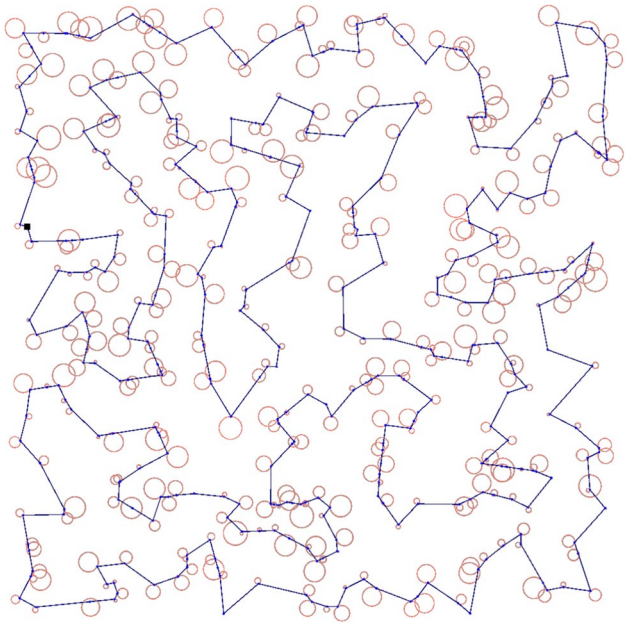


Fig. 4 Visualization of our solution for the rd400rdmRad problem instance.

local optima, completing their evaluations quickly, while others continue to explore the search space for a longer period. Despite this variation, the overall speed-up remains close to linear due to the efficient parallelization of the fitness function. The nearly ideal speed-up is achieved because fitness evaluations, often the most computationally expensive part of the algorithm, can be performed independently without frequent synchronization. Additionally, OpenMP's dynamic scheduling can help distribute the workload more effectively, mitigating minor imbalances. The execution time decreases proportionally with the number of threads, making OpenMP an effective choice for accelerating fitness evaluations in population-based optimization algorithms.

In the Single-Pop-PMA version, a single population is maintained on the master node, while fitness evaluations are distributed among worker threads. This approach ensures that the master node handles population updates and selection while delegating computationally expensive fitness calculations to parallel threads. Compared to a fully parallel population approach, this version tends to achieve results faster because the search dynamics remain more controlled. Since all individuals belong to a shared population, diversity is preserved more effectively, reducing premature convergence risks. Additionally, the master node can make centralized decisions, preventing inefficient searches in isolated subpopulations. Fitness calculations remain parallelized, yielding near-linear speed-up, but the algorithm benefits from faster convergence due to the more efficient exploration-exploitation balance. The synchronization overhead of merging results from multiple populations is eliminated, further enhancing execution efficiency. As a result, this approach not only accelerates fitness evaluation but also improves convergence behavior, leading to quicker solutions without sacrificing performance.

Table 4 gives the details of our execution times for the proposed PMA and MA-CETSP algorithms (we have observed and reported the best execution times of the algorithms). In this Table, the MA-CETSP performed a maximum of 5000 fitness calculations using a single thread, Multi-Pop-PMA performed 16 times more calculations (16 x 5000 = 80,000) using 16 threads, and Single-Pop-PMA performed a maximum of 5000 fitness calculations using 16 threads. Among these, Multi-Pop-PMA conducted the highest number of calculations. However,

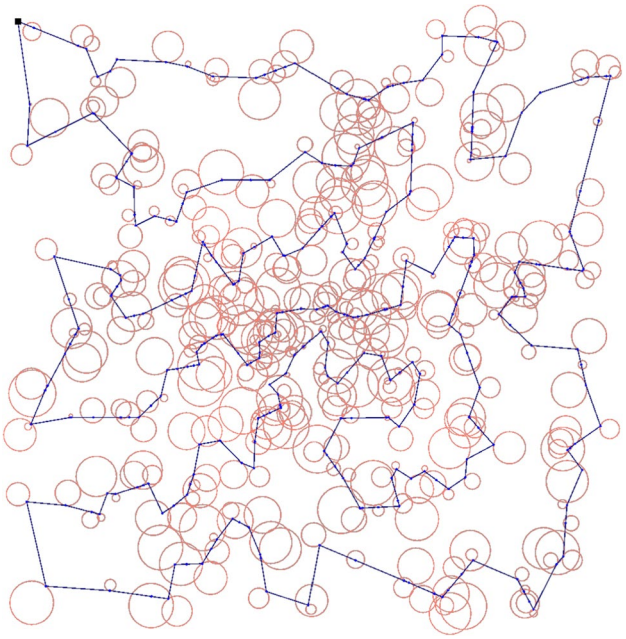


Fig. 5 Visualization of our solution for the team4_400rdmRad problem instance.

the fastest results were achieved by Single-Pop-PMA. Our goal while developing these algorithms was to solve large instances more quickly by leveraging single-population-multiple-thread and multiple-population-multiple-thread approaches to achieve diversification (effective exploration). Multi-Pop-PMA, with its higher number of calculations, ensures a broader exploration of the solution space, while Single-Pop-PMA focuses on speed, delivering results faster. This balance between exploration and efficiency was a key objective in our algorithm design.

Table 5 presents the execution of our Multi-population-PMA algorithm on the bonus1000rdmRad problem instance. There were 16 different populations at the threads. The best values obtained by threads, the total execution times, and the number of iterations (convergence) are presented in this Table. The best time is when the algorithm finds the best solution and continues until it cannot find a better solution, and ends. Total time is the total working time of the algorithm including the iterations without any improvement.

4.3 Comparison with the state-of-the-art algorithms

In this part of our study, we compare the experimental performance results of our algorithms with state-of-the-art algorithms in the literature. In 2019, Wang et al. incorporated the concept of Steiner-zones and designed a rapid three-phase heuristic, SZVNS, based on the Variable Neighborhood Search (VNS) framework to tackle the CETSP [32]. In 2022, Di Placido et al. introduced a powerful Genetic Algorithm (GA) to solve the CETSP [5]. Their approach featured a specially designed crossover operator, resembling a multi-point crossover, along with a mutation operator aimed at enhancing diversification. Additionally, the algorithm incorporated three refinement techniques: a 2-opt operator, SOCP, and a greedy method called 3Alg, inspired by Wang et al. [32]. (lb/ub)alg [22] and HA [26] are the other recent algorithms we have included in our comparisons. The results in Table 3 provide a clear statistical comparison of our proposed parallel memetic algorithm against two state-of-the-art methods. Columns *N* and *OR* respectively represent the target size and overlap ratio of the corresponding instance. In the Appendix, we have reported three of our new best solutions obtained for the problem instances bonus1000rdmRad, rd400rdmRad, and team4_400rdmRad.

We selected 42 of the most difficult, largest and most likely to yield better results in the literature and solved them using our proposed algorithms. Table 3 presents the problem instances we have selected for our study. 21 new best solutions are reported by our study. Bold values given in Table 3 are the new best results for the solution

Table 3 The comparison of algorithms with the sequential version (MA-CETSP). Bold results are the new best solutions reported by our algorithms

Instance	Size (N)	OR	BKS	MA-CETSP	Multi-Pop-PMA	Single-Pop-PMA
bonus1000	1000	12.26%	387.13	389.615	387.978	391.153
bonus1000rdmRad	1000	6.13%	932.07	925.245	**916.539**	918.446
bubbles1	36	11.11%	349.14	349.135	349.135	349.135
bubbles2	76	9.09%	428.28	428.279	428.279	428.279
bubbles3	126	7.69%	529.96	529.955	529.955	529.96
bubbles4	184	6.67%	805.46	807.217	**803.096**	**803.096**
bubbles5	250	5.88%	1038.16	1038.73	**1035.34**	1036.46
bubbles6	324	5.26%	1229.66	1227.4	**1221.02**	1230.12
bubbles7	406	4.76%	1607.31	1584.02	**1573.93**	1585.11
bubbles8	496	4.35%	1946.72	1903.62	**1883.66**	1908.06
bubbles9	594	4.00%	2259.22	2182.5	**2156.77**	2167.29
car_door_50	50	-	4778.91	4778.91	4778.91	4791.59
chaoSingleDep	200	2.84%	1039.61	1039.61	1039.61	1039.61
concentricCircles1	16	15.00%	53.16	53.158	53.158	53.158
concentricCircles2	36	7.50%	153.13	153.167	153.167	156.001
concentricCircles3	60	5.00%	270.04	260.814	**260.784**	261.277
concentricCircles4	104	3.75%	452.64	454.249	**451.87**	452.99
concentricCircles5	148	3.00%	632.99	633.147	**632.977**	**632.977**
d493rdmRad	492	13.44%	134.28	134.226	**134.226**	134.226
dsj1000rdmRad	999	12.47%	624.75	625.211	**624.604**	625.211
kroD100rdmRad	99	4.03%	141.83	141.829	141.829	141.829
lin318rdmRad	317	10.05%	2047.42	2047.11	2047.11	2047.11
pcb442rdmRad	441	9.39%	220	219.933	**219.219**	219.751
rat195rdmRad	194	42.60%	68.22	68.224	68.224	68.224
rd400rdmRad	399	0.99%	1246.69	1241.21	**1238.02**	1238.72
rotatingDiamonds1	20	20.00%	32.39	32.39	32.389	32.389
rotatingDiamonds2	60	5.00%	140.48	140.48	140.48	140.477
rotatingDiamonds3	180	3.33%	380.88	380.882	380.882	380.882
rotatingDiamonds4	320	1.43%	770.66	770.661	770.661	770.661
rotatingDiamonds5	680	1.11%	1510.75	1510.75	1510.75	1510.84
team1_100	100	9.33%	307.34	307.337	307.337	307.337
team1_100rdmRad	100	7.69%	388.54	388.537	388.537	388.537
team2_200	200	20.06%	246.68	246.683	246.683	246.683
team2_200rdmRad	200	5.19%	613.66	613.659	613.659	617.988
team3_300	300	7.02%	464.2	462.169	**461.899**	462.209
team3_300rdmRad	300	23.70%	378.09	378.087	378.087	378.087
team4_400	400	5.01%	685.52	676.562	**671.939**	675.108
team4_400rdmRad	400	2.05%	1000.31	985.135	**983.629**	991.715
team5_499	499	2.00%	700.5	697.821	**694.936**	698.946
team5_499rdmRad	499	20.14%	446.19	446.191	446.191	446.191
team6_500	500	27.06%	225.22	225.216	225.216	225.216
team6_500rdmRad	500	10.02%	620.98	**620.89**	**620.89**	620.911

of the problem instances. See Table 6 for the comparison of our proposed algorithms with state-of-the-art algorithms in the literature.

4.4 Statistical analysis of the results

All experiments were executed on 42 benchmark instances with ten independent runs per instance using 16 threads (see Sect. 4). In this subsection, we summarise the statistical evidence that can be directly derived from

Table 4 The execution times of our proposed parallel memetic algorithms

Instance	MA-CETSP	Multi-Pop-PMA	Single-Pop-PMA
bonus1000	8635.7	35548.5	4257.2
bonus1000rdmRad	1363.1	8268.1	577.4
bubbles1	60.2	286.3	20.9
bubbles2	111.4	580.9	42.9
bubbles3	429.8	1641.3	129.9
bubbles4	1261.0	4029.5	264.8
bubbles5	955.8	5313.4	496.2
bubbles6	2048.5	9604.0	513.0
bubbles7	2612.0	14157.8	1186.3
bubbles8	4487.3	19928.9	1093.9
bubbles9	7295.1	18341.1	1805.8
car_door_50	341.5	822.7	61.9
chaoSingleDep	524.3	1892.3	88.6
concentricCircles1	38.4	127.9	11.3
concentricCircles2	105.4	436.7	33.5
concentricCircles3	307.0	768.7	100.7
concentricCircles4	330.4	2328.8	280.3
concentricCircles5	864.4	3524.9	401.0
d493rdmRad	194.5	975.3	89.4
dsj1000rdmRad	395.6	2721.4	249.241
kroD100rdmRad	184.1	963.1	78.0
lin318rdmRad	291.6	997.0	74.7
pcb442rdmRad	56.9	269.6	145.2
rat195rdmRad	611.9	1805.8	21.4
rd400rdmRad	1543.5	8846.7	546.8
rotatingDiamonds1	37.2	141.8	13.7
rotatingDiamonds2	87.0	471.5	32.4
rotatingDiamonds3	243.5	1400.4	115.0
rotatingDiamonds4	829.4	3403.8	235.2
rotatingDiamonds5	3148.9	10011.6	651.1
team1_100	166.2	953.4	58.6
team1_100rdmRad	89.0	434.8	30.4
team2_200	508.1	2183.9	238.7
team2_200rdmRad	335.1	1987.2	133.4
team3_300	1599.2	7991.7	491.3
team3_300rdmRad	156.7	454.2	37.5
team4_400	4409.4	13375.6	1139.0
team4_400rdmRad	2531.7	6899.1	608.0
team5_499	2447.4	14549.5	1599.7
team5_499rdmRad	140.5	593.5	47.9
team6_500	398.5	3293.4	502.4
team6_500rdmRad	424.0	2879.5	121.4

the per-instance results reported in Table 3 and Table 4, and we highlight additional compact statistics that demonstrate the practical significance of our PMAs.

Firstly, regarding solution quality, Table 3 shows that our algorithms produced a total of 21 new best-known solutions across the 42 instances. In particular, the Multi-Population PMA achieved 19 new best-known solutions while the Single-Population PMA contributed two new best-known solutions. From Table 3 one can therefore directly compute a simple win/tie/loss summary per algorithm versus the Best-Known Solution (BKS): (i) *wins* = number of instances where the algorithm improves the BKS (21 in total, split as above), (ii) *ties* = instances where the algorithm matches the BKS, and (iii) *losses* = instances where the algorithm is worse than the BKS. Reporting this 3-way count gives a compact, interpretable measure of comparative effectiveness.

Table 5 Performance results of Multi-population-PMA algorithm on the bonus1000rdmRad dataset. The best value is indicated with a bold number

Thread id	Best_value	Best_time	Total_time	Number of iterations
1	**916.539**	4418.24	7195.90	889
2	919.035	4380.49	7026.39	936
3	918.489	4625.82	7333.11	946
4	918.603	4589.91	7244.29	971
5	919.678	5355.12	7861.37	1062
6	919.035	5869.82	8150.22	1153
7	924.237	5705.16	7995.80	1183
8	925.301	6555.46	8477.65	1268
9	919.035	6927.46	8579.25	1360
10	919.876	7194.94	8697.33	1380
11	919.411	7334.13	8744.43	1438
12	919.192	7746.03	8869.21	1523
13	922.416	7828.92	8891.63	1542
14	918.604	7800.98	8880.48	1553
15	918.447	7873.55	8892.63	1579
16	919.870	8608.45	9027.17	1884

Secondly, Table 4 provides per-instance runtimes for the sequential MA-CETSP, Multi-Pop-PMA and Single-Pop-PMA. From these data, we report two robust runtime summaries: the *median runtime* across the 42 instances for each algorithm (a robust central tendency measure for skewed time distributions), and the *per-instance speed-up ratio* computed as $t_{\text{sequential}}/t_{\text{parallel}}$. A concise statement of median runtimes and median speed-up (e.g., median($t_{\text{MA-CETSP}}$), median($t_{\text{Single-Pop}}$), median speed-up) succinctly quantifies the practical acceleration achieved by parallelization. Table 4 already shows that the Single-Population PMA is the fastest on the majority of instances, whereas Multi-Pop-PMA performs more total computations but often yields the best-quality solutions (see Table 3).

Thirdly, Table 5 (thread-level breakdown for bonus1000rdmRad) illustrates intra-algorithm variability across threads/populations. From such per-thread data, one can compute the *range* and *standard deviation* of the best values found across threads as well as the *time-to-best* statistics (median and min time when the global best was first discovered). These measures demonstrate both robustness (low variance of best values) and practical usability (how quickly a best solution is obtained).

5 Conclusions and future work

The main objective of this research was to develop scalable and efficient algorithms capable of solving large-scale instances of the CETSP, which is known for its combinatorial complexity and computational challenges. To achieve this goal, we proposed the first two Parallel Memetic Algorithms (PMAs) specifically designed for the CETSP. These include: (i) a Multi-Population PMA that enhances diversity and exploration through distributed crossover and local search operations, and (ii) a Single-Population, Multi-Fitness PMA that accelerates convergence by performing simultaneous fitness evaluations on multi-core architectures. Both strategies effectively utilise OpenMP parallelism to enhance computational efficiency and improve solution quality.

The experimental results obtained from 42 benchmark instances demonstrate that the proposed PMAs significantly outperform existing state-of-the-art algorithms, achieving 21 new best-known solutions. The findings confirm that parallelization not only reduces computation time but also enhances robustness by maintaining population diversity and avoiding premature convergence. Among the two proposed methods, the Multi-Population PMA offers superior exploration capabilities, while the Single-Population PMA achieves faster convergence, making both suitable for different optimisation needs. These results establish new reference benchmarks for

Table 6 The comparison of our proposed algorithms with state-of-the-art algorithms in the literature. The best values obtained by the algorithms are indicated with bold numbers

Instance	BKS	MA-CETSP	Multi-Pop-PMA	Single-Pop-PMA	GA	(lb/ub)alg	SZVNS	HA
bonus1000	387.13	389.615	387.978	391.153	411.85	**387.13**	403.06	408.44
bonus1000rdmRad	932.07	925.245	**916.539**	918.446	932.07	955.41	938.27	1001.10
bubbles1	349.14	349.135	349.135	349.135	349.13	349.13	349.14	349.14
bubbles2	428.28	428.279	428.279	428.279	428.28	428.28	428.28	428.28
bubbles3	529.96	529.955	529.955	529.96	529.96	529.96	532.21	532.28
bubbles4	805.46	807.217	**803.096**	**803.096**	805.46	805.56	825.33	832.27
bubbles5	1038.16	1038.73	**1035.34**	1036.46	1038.16	1061.64	1073.43	1067.96
bubbles6	1229.66	1227.4	**1221.02**	1230.12	1229.66	1313.02	1263.68	1394.14
bubbles7	1607.31	1584.02	**1573.93**	1585.11	1607.31	1650.04	1639.33	1735.38
bubbles8	1946.72	1903.62	**1883.66**	1908.06	1946.72	2021.27	1972.99	2120.98
bubbles9	2259.22	2182.5	**2156.77**	2167.29	2259.22	2413.31	2330.31	2456.27
chaoSingleDep	1039.61	1039.61	1039.61	1039.61	1039.61	1039.61	1039.63	1042.82
concentricCircles1	53.16	53.158	53.158	53.158	53.16	53.16	53.16	53.16
concentricCircles2	153.13	153.167	153.167	156.001	153.13	154.81	154.88	153.13
concentricCircles3	270.04	260.814	**260.784**	261.277	270.04	272.69	272.49	271.08
concentricCircles4	452.64	454.249	**451.87**	452.99	452.64	466.52	461.36	455.62
concentricCircles5	632.99	633.147	**632.977**	**632.977**	632.99	659.36	647.84	647.64
d493rdmRad	134.28	134.226	**134.226**	**134.226**	134.28	134.74	135.02	141.34
dsj1000rdmRad	624.75	625.211	**624.604**	**624.604**	624.75	625.92	625.25	659.64
kroD100rdmRad	141.83	141.829	141.829	141.829	141.83	142.39	141.83	141.84
lin318rdmRad	2047.42	2047.11	2047.11	2047.11	2047.42	2055.77	2082.25	2079.49
pcb442rdmRad	220.0	219.933	**219.219**	219.751	220.00	220.44	221.16	234.15
rat195rdmRad	68.22	68.224	68.224	68.224	68.22	68.22	68.22	68.22
rd400rdmRad	1246.69	1241.21	**1238.02**	1238.72	1246.69	1305.46	1257.73	1252.22
rotatingDiamonds1	32.39	32.39	32.389	32.389	32.29	32.29	32.39	32.29
rotatingDiamonds2	140.48	140.48	140.48	140.477	140.48	140.48	140.48	140.48
rotatingDiamonds3	380.88	380.882	380.882	380.882	380.88	380.89	380.89	382.50
rotatingDiamonds4	770.66	770.661	770.661	770.661	770.66	772.00	770.68	777.05
rotatingDiamonds5	1510.75	1510.75	1510.75	1510.84	1510.75	1531.74	1510.88	1530.31
team1_100	307.34	307.337	307.337	307.337	307.34	307.34	307.34	307.34
team1_100rdmRad	388.54	388.537	388.537	388.537	388.54	388.54	388.54	390.23
team2_200	246.68	246.683	246.683	246.683	246.68	246.68	246.69	247.48
team2_200rdmRad	613.66	613.659	613.659	617.988	613.66	616.82	626.90	624.79
team3_300	464.2	462.169	**461.899**	462.209	464.20	476.43	465.80	466.12
team3_300rdmRad	378.09	378.087	378.087	378.087	378.09	378.51	379.84	382.16
team4_400	685.52	676.562	**671.939**	675.108	685.52	702.69	698.05	686.76
team4_400rdmRad	1000.31	985.135	**983.629**	991.715	1000.31	1025.76	1006.71	1020.16
team5_499	700.5	697.821	**694.936**	698.946	700.50	708.45	703.38	711.14
team5_499rdmRad	446.19	446.191	446.191	446.191	446.19	446.51	703.38	458.35
team6_500	225.22	225.216	225.216	225.216	225.22	225.22	226.18	227.50
team6_500rdmRad	620.98	**620.89**	**620.89**	620.911	620.98	626.18	621.99	672.36

large-scale CETSP instances and highlight the effectiveness of parallel memetic approaches in complex optimization scenarios.

From a practical standpoint, the study recommends adopting the proposed PMAs for real-world applications such as UAV mission planning, robotic navigation, and sensor network routing, where both accuracy and computation time are critical. Moreover, increasing the number of processing threads and fine-tuning algorithmic parameters can yield even greater performance improvements.

Future research directions include integrating machine learning and deep reinforcement learning models to guide adaptive parameter tuning and dynamic solution selection. Additionally, developing hybrid frameworks that combine PMAs with surrogate modeling or neural-based prediction techniques could further reduce

computational costs while preserving solution quality. Expanding the algorithms to handle multi-objective and dynamic variants of the CETSP represents another promising avenue. Overall, this study provides a strong foundation for future exploration of parallel and intelligent metaheuristics in large-scale combinatorial optimization.

Appendix A

Our multi-population memetic algorithm obtains new best solutions, as given below. The three numbers given in a sequence are the ID of the location (bold values) and the x and y values. The salesman visits the locations in the given order of the locations in these results.

bonus1000rdmRad

0 80 20 **330** 86.5566 11.8539 **172** 90.7637 12.2455 **63** 90.9358 17.7054 **219** 90.9825 19.1884 **34** 91.3187 29.8518 **186** 87.5369 36.2078 **98** 81.4316 35.1699 **84** 76.6481 34.3563 **118** 70.6994 40.7332 **210** 76.1127 46.2351 **75** 77.0264 47.2241 **312** 78.576 50.56 **325** 79.1519 52.0499 **142** 79.977 54.1836 **287** 83.1891 53.9861 **322** 85.9401 53.8169 **38** 90.7003 56.0008 **179** 94.8983 57.9267 **166** 88.7353 66.9723 **189** 88.6372 66.9682 **79** 88.5598 66.965 **122** 84.6051 65.3534 **140** 82.7527 63.9924 **236** 81.4609 63.0432 **123** 76.4021 59.3287 **206** 72.7405 55.0257 **228** 70.9075 58.0721 **260** 67.2116 57.6439 **319** 67.2116 57.6439 **73** 67.2116 57.6439 **226** 65.7154 62.5046 **24** 64.7845 63.3353 **88** 63.004 64.9249 **308** 63.144 65.1525 **248** 67.0069 71.444 **294** 69.3046 70.1652 **56** 71.7481 70.1988 **306** 71.8317 70.2468 **29** 72.9654 73.8814 **203** 73.1451 74.4582 **156** 75.0589 80.5908 **252** 76.8387 86.2949 **225** 80.7641 86.4882 **247** 82.7839 87.1576 **10** 92.036 86.7981 **117** 85.9916 91.3081 **332** 80.336 95.3741 **204** 77.043 93.9657 **177** 75.7952 93.4319 **293** 73.9217 92.6305 **278** 72.3503 90.2952 **105** 69.8758 91.9987 **195** 67.8164 93.4166 **13** 65.8824 92.6489 **65** 64.4899 91.1341 **216** 65.7601 88.8378 **200** 66.4775 87.5409 **168** 68.203 84.4204 **314** 62.0187 82.3105 **6** 62.3903 79.6576 **231** 59.3329 75.8794 **272** 58.7075 75.0875 **292** 58.6225 74.9802 **40** 55.6214 72.0502 **39** 53.6218 70.9169 **31** 51.3648 73.5019 **33** 50.9948 74.6159 **273** 50.3417 76.5821 **264** 52.0672 77.3707 **124** 55.4687 81.0995 **51** 54.7211 82.7277 **151** 54.0137 84.268 **91** 54.3885 85.2355 **96** 54.9578 86.7056 **41** 56.1211 89.7121 **78** 55.6161 90.5303 **261** 53.9213 93.2007 **217** 53.7234 97.5734 **127** 53.6561 99.069 **9** 53.5449 99.0467 **194** 48.5987 98.0587 **138** 44.6051 97.2125 **244** 45.9887 94.7338 **254** 46.3987 93.9998 **114** 46.4437 92.8034 **153** 46.4437 92.8034 **258** 44.3381 93.4329 **80** 43.0376 93.8215 **201** 40.7283 95.9125 **259** 39.8683 96.6915 **67** 38.7664 97.3907 **50** 37.072 98.4654 **215** 36.8445 97.4931 **251** 37.0961 97.1855 **8** 37.2705 96.3149 **165** 37.3239 96.0473 **144** 37.7979 92.6288 **54** 37.9942 91.2154 **193** 38.3378 88.7383 **145** 38.5845 86.9608 **89** 39.093 84.9547 **198** 39.8944 82.6584 **288** 39.9406 82.5256 **74** 44.6674 81.5553 **199** 44.3153 79.8901 **28** 44.2815 79.8483 **108** 43.8645 79.3322 **53** 43.3727 74.1079 **173** 45.4826 70.2488 **87** 45.4825 70.2488 **47** 43.8578 69.597 **83** 41.8063 68.774 **245** 39.713 67.9341 **150** 39.668 67.9161 **135** 38.5867 65.2104 **235** 39.3985 63.6855 **17** 39.9102 62.7245 **37** 39.9881 62.6431 **249** 41.8526 60.6974 **197** 44.5904 60.8727 **313** 48.5 56.7324 **102** 48.8365 57.6841 **271** 49.9038 60.7012 **326** 52.5843 64.8223 **253** 52.6759 64.7277 **30** 53.7248 63.6417 **106** 55.1175 59.1741 **276** 55.214 58.864 **70** 56.0484 56.1848 **72** 56.9405 55.2339 **146** 58.3386 53.7431 **25** 60.4683 49.9105 **290** 61.2064 48.5827 **223** 61.6967 47.701 **52** 61.8003 47.5149 **211** 59.3774 41.8092 **131** 60.204 38.065 **263** 57.1993 35.1327 **1** 55.247 33.2271 **283** 54.9696 32.9561 **233** 54.6022 32.5973 **18** 53.0466 31.0787 **298** 51.7192 29.7829 **60** 51.719 29.7834 **266** 51.0347 31.2861 **7** 49.4851 34.691 **316** 48.3701 37.1406 **76** 48.2408 39.2394 **307** 50.2639 42.5788 **299** 51.813 45.1358 **21** 51.6742 45.3516 **175** 49.9701 48.0066 **183** 45.0163 45.8953 **57** 43.1402 45.0953 **66** 39.9294 49.5703 **81** 39.3226 51.4889 **137** 38.6526 53.6075 **300** 34.9305 52.9842 **213** 32.8264 53.5646 **12** 34.3125 58.5994 **176** 32.5594 60.1521 **48** 31.7204 62.708 **143** 29.5168 65.0193 **132** 30.2337 67.0637 **181** 30.6433 68.2315 **209** 31.4695 70.5891 **214** 29.5089 74.9804 **167** 30.2069 76.4945 **93** 30.5572 77.2546 **44** 30.5924 77.3309 **237** 30.6184 77.3873 **275** 32.2256 80.8727 **97** 32.2256 80.8727 **110** 30.0902 82.1908 **61** 29.815 83.5589 **94** 29.1552 86.8344 **267** 28.301 92.4665 **26** 27.2202 92.7706 **257** 25.3817 93.2884 **154** 25.3266 97.4884 **11** 25.3196 98.0474 **159** 25.3056 99.1561 **297** 20.9622 98.2785 **85** 18.8188 97.8453 **4** 18.3886 97.7585 **262** 18.286 97.7378 **315** 19.2901 94.3741 **243** 19.696 93.0153 **284** 20.6426 90.7477 **16** 19.9856 89.3323 **69** 19.6728 89.081 **239** 18.3264 87.9986 **133** 17.9631 87.7067 **32** 17.2752 88.1633 **5** 16.3413 91.1073 **163** 15.1294

92.093 **141** 14.0416 92.9778 **329** 12.9146 93.8946 **49** 11.5184 95.0301 **160** 7.33908 98.5133 **334** 1.87317 98.073 **295** 2.81273 95.0589 **232** 3.61988 92.4699 **327** 4.07098 92.0453 **77** 5.81777 90.909 **274** 5.9992 89.5543 **246** 7.15963 88.664 **221** 7.52108 88.387 **220** 8.47468 87.6556 **279** 9.0486 87.2155 **196** 3.64282 86.2568 **62** 3.42083 85.6976 **148** 3.36907 85.5674 **158** 2.34643 82.9883 **320** 1.51845 80.9 **116** 8.57472 77.8733 **161** 8.52021 77.7616 **128** 8.26264 77.2347 **336** 5.78163 75.639 **111** 3.98629 73.3055 **238** 2.19933 70.9829 **324** 1.83834 69.3035 **229** 3.62586 68.5109 **100** 5.05768 67.8763 **86** 5.57785 64.5176 **256** 7.96263 64.5437 **301** 10.6051 64.6661 **187** 16.3171 64.9314 **27** 15.5874 65.9438 **171** 14.901 66.8959 **147** 14.7104 68.3707 **182** 14.0283 71.7914 **59** 14.2888 72.3273 **180** 14.7779 73.3326 **270** 16.0623 75.9709 **90** 18.0342 77.3275 **321** 21.0498 76.2576 **164** 21.3587 76.7598 **202** 23.4736 80.1935 **192** 23.8554 75.7036 **304** 23.405 73.9585 **35** 23.1229 72.8662 **99** 23.3189 69.8163 **119** 24.1838 68.8617 **291** 22.8268 64.2007 **305** 22.7855 62.5652 **139** 24.8653 60.8474 **250** 24.3975 58.8688 **20** 25.5396 56.6368 **331** 21.8546 56.0663 **318** 19.8574 54.9883 **125** 15.4322 55.0059 **328** 13.3031 55.0149 **317** 10.9146 50.7261 **68** 10.7747 50.1639 **191** 10.5802 49.3826 **289** 10.1469 48.8117 **310** 9.03047 49.5764 **113** 6.71984 51.1585 **19** 5.60235 51.4761 **82** 4.20533 51.8732 **42** 2.52869 52.3494 **242** 2.57064 52.1012 **15** 2.78185 50 **46** 3.48423 46.6964 **120** 4.0164 43.5487 **14** 6.99138 42.8076 **103** 9.71622 42.1289 **335** 10.2031 42.2807 **43** 10.5824 42.399 **240** 13.4298 43.2852 **169** 16.8479 44.3489 **255** 18.7617 48.0745 **178** 19.498 48.0405 **309** 22.7337 47.8921 **311** 26.2655 47.315 **188** 26.0146 45.8487 **162** 25.703 44.026 **23** 24.7545 38.4726 **45** 24.648 37.8506 **296** 27.6669 34.1889 **101** 29.575 34.9282 **190** 31.3399 35.8579 **55** 31.828 37.1947 **277** 31.8305 37.2014 **265** 35.0197 40.1379 **2** 35.8726 39.8009 **170** 38.3352 38.8272 **157** 40.6599 34.3411 **207** 39.5582 32.0356 **285** 37.3199 29.8029 **281** 34.0983 26.3757 **268** 30.5271 26.7011 **282** 27.3694 26.9891 **130** 24.8925 27.2304 **286** 16.7647 28.0226 **333** 11.8954 27.4291 **112** 8.14813 26.6274 **269** 6.99894 26.3818 **184** 6.38162 26.25 **121** 5.71825 24.0699 **95** 5.43097 23.1262 **71** 4.0038 18.4343 **224** 4.74656 17.1954 **155** 5.38411 16.1319 **234** 6.12368 14.8984 **208** 6.85283 13.6822 **302** 9.36636 9.18482 **174** 14.7405 11.9614 **3** 17.1355 13.8785 **136** 19.9126 16.1016 **222** 23.2827 18.7987 **241** 23.7752 19.67 **92** 25.8706 20.5735 **104** 25.9365 20.6019 **22** 28.071 20.0101 **218** 29.5037 19.6127 **230** 30.8471 14.0646 **152** 31.0226 14.1905 **134** 31.19 14.3105 **129** 31.3437 14.4208 **303** 36.4956 18.1076 **126** 37.8669 17.986 **115** 39.0775 17.8787 **227** 43.8945 17.4517 **185** 42.8259 6.20935 **107** 42.8904 6.20946 **36** 61.5368 6.24437 **64** 63.0914 12.0236 **212** 64.6376 17.7714 **205** 64.9267 18.848 **323** 65.2928 20.211 **109** 65.5306 20.1366 **280** 65.7711 20.0613 **149** 70.7546 18.5036 **58** 76.3015 15.7938

rd400rdmRad

0 2.28315 35.5085 **278** 1.05081 35.1762 **334** 3.38472 28.1946 **267** 3.40298 27.3513 **301** 3.02238 26.0799 **81** 2.83969 25.4696 **178** 2.40925 24.0317 **233** 3.91614 21.2607 **342** 1.07765 19.546 **375** 2.13573 16.3778 **317** 0.956223 12.343 **190** 2.6081 10.5561 **166** 4.52291 8.4848 **24** 2.95476 5.74318 **156** 1.61525 3.40121 **27** 4.95042 3.42565 **187** 8.56947 3.45215 **355** 8.86708 3.45419 **89** 10.7511 3.83634 **383** 12.7025 4.23217 **297** 19.7234 0.339819 **152** 21.5845 1.5621 **79** 22.5253 2.17997 **345** 24.209 3.28579 **149** 26.9396 5.07906 **182** 32.7821 2.42786 **194** 37.1979 7.37729 **200** 38.3156 8.63012 **172** 42.8822 4.72319 **285** 47.3075 2.41924 **145** 47.944 2.08787 **77** 49.5064 7.19296 **47** 51.3025 6.49578 **380** 52.574 6.00223 **351** 57.2127 6.52863 **17** 56.9136 2.72222 **294** 56.8515 1.93285 **223** 60.5447 1.0458 **174** 61.3968 0.841118 **207** 64.2812 3.99877 **134** 68.2358 8.32818 **179** 70.5399 5.57809 **128** 73.318 6.11358 **256** 74.1984 6.28329 **350** 77.0672 10.2647 **66** 77.7678 13.9519 **75** 78.1641 16.0372 **231** 78.167 17.3217 **88** 77.4846 17.3414 **225** 76.7901 17.3615 **319** 75.3765 17.4023 **57** 76.8579 22.2251 **312** 82.2875 22.5421 **169** 86.4839 17.7488 **138** 87.9134 14.8349 **188** 89.1776 12.2581 **198** 89.2829 11.0214 **73** 91.6803 10.4529 **266** 89.848 1.66253 **337** 95.7104 2.65184 **360** 97.9024 5.19582 **281** 98.2082 7.80622 **291** 97.5637 17.1788 **358** 98.212 23.5677 **272** 98.2623 24.4158 **80** 96.9947 23.3332 **175** 96.814 23.1789 **43** 93.1171 20.0217 **132** 90.5919 21.1678 **143** 88.9384 25.52 **283** 88.0653 29.8735 **164** 85.0732 30.891 **31** 82.2343 29.9033 **99** 80.1224 32.5847 **184** 77.6749 29.3578 **292** 75.0659 32.5309 **299** 75.4877 34.8307 **320** 75.4964 34.8782 **341** 79.9203 36.9573 **307** 80.1688 38.0695 **356** 75.7297 40.3666 **137** 73.2103 41.0621 **93** 71.8874 43.3454 **98** 70.2267 44.0184 **105** 70.2025 46.7585 **372** 72.4497 47.219 **259** 74.8577 48.0715 **70** 78.5306 48.1115 **210** 79.4725 46.118 **280** 79.7009 44.306 **221** 81.5401 43.9912 **163** 85.4241 43.3262 **208** 88.1211 43.0016 **26** 91.2058 42.6304 **120** 92.6019 41.3562 **125** 95.8414 38.3995 **362** 95.2076 40.9595 **41** 94.645 43.2317 **11** 94.0642 43.7326 **310** 91.849 45.6431 **59** 90.1061 47.1462 **286** 87.0588 48.5831 **168** 88.4507 55.9103 **211** 95.8473 58.6233 **374**

88.7845 66.5988 **251** 93.1644 68.8818 **29** 96.9254 72.1834 **232** 96.9693 72.8792 **119** 95.067 73.8462 **180** 95.4353 79.1202 **139** 97.9682 82.1952 **359** 92.882 87.8739 **379** 93.4884 89.7034 **15** 95.5543 95.9376 **347** 91.0464 95.2769 **197** 88.3669 95.7904 **3** 87.6487 94.537 **87** 83.9935 88.158 **129** 83.9993 86.4809 **316** 81.1202 88.404 **373** 78.2595 90.6856 **303** 76.994 92.3176 **185** 76.9534 92.3699 **110** 75.6104 99.6356 **376** 73.6315 97.9512 **261** 68.6109 94.5538 **318** 66.9293 96.8196 **126** 65.7687 97.6642 **10** 60.0862 95.473 **94** 57.9889 93.9608 **349** 55.0417 95.4513 **235** 53.6783 98.5289 **214** 51.9089 98.5095 **55** 49.3924 97.7132 **30** 48.4561 96.5741 **22** 40.5356 94.7059 **181** 34.7965 99.6552 **118** 34.1684 93.3651 **45** 33.8013 92.0765 **242** 33.3641 90.5422 **64** 32.5108 87.4399 **44** 28.6889 89.9688 **117** 28.7023 90.6201 **196** 28.7305 91.9877 **33** 27.1871 93.5295 **260** 23.2774 94.8953 **68** 18.3524 91.3375 **311** 13.8308 91.8627 **268** 15.444 94.7939 **147** 16.5982 94.5775 **176** 17.1946 96.2791 **34** 16.7131 97.2795 **113** 12.6191 97.6904 **365** 3.66886 98.5886 **332** 0.978963 97.1183 **203** 2.85047 93.475 **220** 2.85049 93.475 **328** 3.14673 92.5676 **46** 2.55969 89.2175 **54** 2.37373 88.1565 **130** 1.53522 83.3715 **6** 8.27945 83.2188 **144** 8.38663 82.3604 **277** 4.36742 74.7103 **92** 1.08661 71.0138 **230** 2.10692 66.8951 **23** 2.68536 62.5481 **18** 5.55507 62.1012 **12** 7.42569 61.8099 **370** 8.55039 64.1966 **287** 9.55079 65.8313 **40** 14.0854 70.1817 **377** 16.3806 69.822 **290** 23.4331 68.7167 **20** 23.9097 69.6254 **14** 24.1809 70.1424 **28** 25.4778 73.1156 **112** 20.8885 75.5769 **186** 17.6449 77.7666 **101** 17.5071 77.8597 **153** 17.3463 80.7618 **60** 19.6261 82.1557 **7** 23.0542 84.2518 **305** 24.36 80.3253 **327** 25.6351 80.3067 **65** 26.9672 80.0299 **71** 29.8819 79.7819 **382** 34.8274 80.2792 **265** 37.4052 81.4764 **309** 35.3848 83.7201 **95** 37.844 86.0921 **336** 40.7082 85.9841 **107** 43.1972 85.8903 **136** 45.6225 87.1748 **58** 48.7067 88.8081 **348** 50.2582 90.9605 **243** 51.2197 90.4901 **331** 53.6053 89.3232 **56** 53.4399 87.4138 **282** 51.3484 83.4609 **141** 52.3911 82.0587 **104** 50.4213 80.4532 **330** 48.3117 81.2387 **314** 47.5145 81.5356 **151** 46.1025 81.312 **296** 45.3075 81.1861 **124** 43.939 81.7068 **300** 43.0968 81.5758 **240** 42.5777 81.495 **289** 42.6094 75.2944 **325** 46.1241 71.522 **150** 46.4658 71.1552 **262** 43.9161 68.1335 **52** 43.707 67.4537 **264** 45.1498 66.2708 **193** 48.7451 63.3232 **344** 51.7529 64.9836 **86** 52.6642 66.9763 **353** 56.3 62.594 **21** 58.4154 61.3435 **246** 59.2412 60.855 **62** 60.3208 60.2168 **205** 63.8464 60.1989 **217** 64.6165 60.7749 **53** 69.2217 64.2197 **171** 67.0092 66.2557 **90** 66.3076 67.7172 **97** 65.7169 68.9477 **102** 61.6056 68.3992 **340** 61.2402 71.3825 **346** 61.1 72.5276 **288** 60.8316 74.719 **371** 60.6076 76.5472 **133** 60.51 77.8837 **2** 60.345 80.145 **116** 61.2026 81.3531 **199** 63.7222 84.9029 **254** 66.9549 81.4762 **368** 73.779 81.5522 **202** 75.6299 80.6822 **270** 77.7271 79.5332 **5** 79.5146 79.9829 **244** 82.5202 80.7392 **255** 85.1915 82.0427 **339** 89.0944 77.1647 **49** 86.5775 76.1356 **13** 86.0042 75.9621 **135** 84.1 75.6932 **249** 81.7534 75.3618 **269** 80.8584 75.2354 **216** 77.5042 74.7617 **218** 78.2587 74.0974 **363** 81.8392 70.9452 **323** 79.022 68.5921 **239** 81.8521 63.0256 **213** 84.9033 62.0346 **258** 83.0253 59.9936 **364** 82.3866 58.2277 **35** 80.1238 54.7874 **313** 76.3914 55.1147 **76** 76.0321 55.1462 **100** 75.2826 57.8137 **263** 70.9658 57.1406 **19** 69.6638 56.9376 **354** 69.1123 55.8789 **159** 67.5464 55.4683 **357** 62.4505 54.132 **140** 57.9235 54.0503 **279** 54.6606 52.4469 **338** 54.5409 45.6012 **106** 61.2356 41.4911 **253** 59.8579 36.7292 **236** 56.6976 37.0012 **103** 56.9281 36.0094 **38** 56.4248 35.5169 **161** 56.2176 33.1707 **51** 56.1271 32.1469 **366** 60.8848 27.4244 **91** 63.85 20.2408 **209** 66.9242 15.4813 **69** 66.744 14.9572 **384** 59.4232 15.6042 **324** 56.673 17.7983 **183** 54.7198 20.4435 **8** 48.4466 19.9075 **321** 48.6675 19.2838 **142** 49.6155 16.6069 **85** 49.4483 16.3857 **308** 44.0069 14.0342 **167** 41.3133 18.526 **219** 40.1185 18.3152 **248** 36.0857 17.6037 **302** 36.0136 21.9186 **335** 38.2927 22.6284 **173** 42.5145 23.9433 **122** 47.3463 25.4482 **36** 45.1058 30.5049 **229** 49.101 32.8664 **127** 46.1294 41.172 **238** 45.6101 41.4973 **154** 38.4881 45.9592 **42** 38.2033 48.637 **245** 37.5521 49.4226 **162** 41.3001 52.5511 **4** 43.99 55.4638 **1** 43.5211 58.6806 **191** 42.0355 58.7072 **9** 40.774 60.4382 **131** 35.992 66.9993 **109** 32.705 61.5429 **293** 32.5592 60.8066 **212** 32.5386 60.7021 **37** 31.438 55.1433 **284** 31.1189 53.5317 **32** 31.6979 46.7395 **275** 31.1477 43.9663 **333** 30.4903 43.3473 **247** 33.0728 38.9222 **227** 34.3632 38.6962 **115** 36.7328 32.1985 **361** 37.003 31.4575 **74** 36.0548 29.1728 **84** 32.0893 29.7545 **222** 30.1107 28.0234 **369** 27.5016 25.7407 **96** 26.8171 25.1417 **155** 30.2505 22.1924 **250** 27.0665 20.683 **215** 27.0325 19.6461 **276** 26.9676 17.6655 **304** 25.959 17.5539 **108** 24.4368 14.4023 **378** 24.2939 13.3023 **352** 20.9719 9.92522 **61** 17.2481 10.7614 **78** 15.4303 11.1696 **367** 12.6503 12.4335 **329** 16.9774 17.2574 **67** 15.9285 17.7325 **39** 15.2652 18.033 **326** 11.596 19.6954 **146** 13.6067 22.6582 **121** 15.0434 23.4303 **177** 15.8801 25.0111 **241** 19.4826 28.3186 **82** 19.3392 29.2428 **234** 20.1031 30.0591 **48** 22.932 33.0823 **228** 23.2311 33.402 **158** 25.2312 33.733 **298** 24.9821 40.0637 **170** 24.8997 42.3375 **204** 23.4583 46.9625 **274** 22.9908 48.4626 **192**

20.8439 48.208 **271** 19.0863 52.2722 **111** 18.9481 54.04 **206** 22.1901 54.5665 **322** 22.6647 55.6507 **123** 23.8225 58.296 **114** 24.3633 59.5315 **237** 24.6032 60.0796 **315** 19.0043 60.8711 **25** 18.8845 60.8881 **160** 17.6266 61.0659 **157** 13.4589 58.1405 **165** 11.5553 58.0576 **306** 12.0843 55.804 **16** 12.5785 53.6987 **63** 11.9766 51.1615 **195** 11.3981 49.6846 **257** 8.75747 52.0168 **295** 3.81346 53.4262 **83** 2.62198 50.9889 **381** 7.41075 42.7903 **224** 9.29499 43.1118 **343** 12.0881 43.0405 **201** 13.4937 42.2797 **148** 15.1225 42.9563 **226** 16.6616 40.5355 **252** 17.2497 36.8637 **273** 11.4164 37.6516 **72** 9.66427 37.7825 **189** 8.70548 37.8541 **50** 2.96254 37.9476

team4_400rdmRad

0 0.0 0.0 **28** 2.6705 1.08305 **39** 10.0485 4.07505 **149** 11.4301 4.63573 **132** 13.3302 9.28876 **130** 15.4464 8.35831 **155** 16.5844 6.40766 **167** 24.7598 6.83519 **112** 28.3403 7.57012 **235** 32.6068 9.2337 **20** 39.8523 9.48126 **121** 41.2441 8.95591 **313** 43.1046 8.2537 **72** 44.7368 7.63764 **102** 46.3411 7.03211 **53** 53.3572 11.2448 **129** 54.9694 12.2129 **114** 57.7807 12.9943 **38** 57.9385 13.0381 **176** 58.1052 13.0844 **265** 58.5708 13.2136 **237** 61.8189 10.977 **50** 62.6403 10.7722 **111** 64.4621 10.3176 **128** 67.6083 9.86524 **41** 70.8951 5.53895 **101** 70.9075 2.44053 **45** 76.9554 2.65381 **151** 79.7153 3.35811 **109** 80.123 3.46202 **6** 76.1636 11.8434 **158** 75.9366 17.7115 **123** 75.9238 18.0459 **93** 75.908 18.4559 **37** 75.7706 22.0265 **204** 75.6391 23.1514 **14** 81.5085 22.5999 **107** 84.1924 15.3284 **25** 87.3167 11.9601 **146** 94.1987 9.73709 **117** 95.2465 9.39869 **33** 98.3744 9.19232 **42** 99.0433 9.14822 **48** 97.0877 16.6598 **88** 94.0189 28.4498 **184** 92.2496 35.2451 **241** 92.5072 35.95 **80** 94.5493 41.5388 **104** 94.2895 41.832 **9** 91.8468 41.4744 **169** 87.1271 40.7837 **222** 86.6562 40.7148 **195** 83.7877 40.295 **71** 83.0529 42.1187 **322** 83.9963 43.8379 **177** 84.4959 44.7484 **285** 84.7737 46.1248 **216** 83.2965 46.1624 **306** 82.5758 46.1807 **234** 80.3825 49.0893 **126** 82.2788 50.9016 **242** 85.8648 54.3291 **44** 93.575 54.9553 **113** 94.1693 57.5475 **193** 94.9531 60.9675 **70** 97.3945 67.8305 **116** 93.1549 70.8056 **29** 92.2962 71.5115 **168** 91.8719 71.8604 **22** 94.3054 75.1963 **12** 91.7786 81.1052 **7** 91.5627 84.0828 **124** 91.1561 89.6804 **85** 90.0582 90.9595 **91** 90.0253 91.024 **56** 86.1865 98.5691 **49** 85.4925 97.2715 **127** 84.5501 95.5095 **61** 83.4795 93.5077 **36** 76.0694 90.3525 **19** 74.1677 92.638 **17** 71.8887 95.3776 **118** 70.952 96.5033 **43** 69.6335 95.9382 **157** 69.5104 95.8855 **115** 65.3241 94.0925 **30** 64.5082 93.7496 **31** 56.4349 90.3574 **173** 50.3902 87.8175 **120** 48.5665 97.8484 **23** 41.5794 95.5777 **66** 40.6375 89.9813 **140** 40.5324 89.3573 **55** 38.6403 84.8033 **171** 37.053 83.9652 **211** 35.9154 83.3646 **63** 33.2918 86.8554 **8** 32.3421 88.4106 **125** 30.8253 90.8943 **199** 28.2231 90.9868 **46** 26.2672 91.0562 **106** 25.1498 96.7879 **5** 22.3629 96.3977 **134** 15.3521 95.416 **77** 4.6834 94.6757 **2** 1.46798 80.5395 **69** 7.98078 81.1309 **62** 12.3432 82.2719 **291** 13.6412 82.6113 **156** 14.9477 79.1248 **13** 16.3256 79.2189 **283** 19.96 78.0792 **40** 21.0092 77.7504 **257** 21.8269 77.4941 **95** 24.1747 82.4378 **35** 26.0512 81.2146 **326** 28.2747 79.3627 **229** 28.8815 77.56 **310** 29.4208 75.957 **161** 30.3754 70.9189 **24** 30.3794 70.898 **131** 33.8212 70.2814 **4** 37.4286 74.0243 **110** 39.4936 74.8778 **263** 39.6452 70.4475 **227** 40.213 66.5247 **202** 41.1665 66.0835 **319** 42.1556 65.6258 **170** 43.9973 64.7736 **250** 43.9554 64.5782 **254** 43.898 64.3109 **214** 43.5109 62.5079 **268** 42.1236 60.8732 **83** 44.8458 58.9963 **249** 46.2916 55.8864 **253** 46.3064 55.8547 **256** 47.7823 54.5028 **166** 48.7763 53.5924 **226** 50.6005 52.0112 **295** 50.6683 52.0153 **264** 51.8039 53.8551 **316** 51.3081 58.0878 **79** 51.7339 59.4588 **271** 53.4138 60.6701 **208** 54.9835 58.8655 **207** 57.8321 55.5906 **231** 60.7191 56.0658 **82** 62.1987 60.0319 **188** 62.2644 60.1176 **223** 63.8025 58.8513 **182** 65.9093 57.1169 **98** 67.1904 59.0565 **139** 66.2373 64.2291 **135** 66.2306 64.2388 **221** 63.6895 67.9154 **305** 61.9023 66.395 **178** 61.5384 66.4011 **16** 55.7508 66.4969 **282** 53.9774 66.5262 **243** 51.1659 72.0303 **154** 51.622 72.6825 **224** 52.9946 74.6455 **323** 54.3518 76.5866 **143** 57.1615 81.5994 **96** 57.5772 81.5792 **67** 60.2073 81.4508 **273** 63.7453 81.2779 **230** 63.4803 80.3232 **150** 62.4392 76.5734 **175** 63.2673 75.0795 **320** 65.7235 76.6375 **97** 67.9391 75.4203 **68** 69.9493 76.8937 **144** 71.9782 78.3809 **160** 74.7848 76.4307 **52** 75.0209 73.4089 **47** 79.5046 70.9848 **190** 76.9678 68.0912 **321** 75.4151 66.3201 **213** 77.185 61.3826 **138** 75.5737 56.9427 **294** 75.0774 55.5753 **210** 73.9618 52.5014 **297** 72.0265 50.7981 **185** 72.6536 49.0705 **212** 73.2344 47.4713 **164** 73.9189 45.5853 **327** 75.5775 41.0159 **244** 75.8346 40.3073 **65** 76.2573 39.1427 **288** 76.1662 36.3819 **108** 74.4326 36.2561 **92** 73.02 36.1539 **274** 72.6735 36.7307 **186** 72.0378 37.7889 **201** 69.3226 42.3096 **311** 64.9051 40.1207 **236** 64.3367 44.4095 **233** 63.9226 47.5333 **58** 61.6317 48.2162 **64** 61.2836 48.3198 **304** 58.318 49.2037 **279** 56.7865 49.4939 **220** 56.077 49.6283 **266** 55.0114 49.2854 **3** 54.5353 49.1321 **281** 53.1559 48.6882 **219** 51.7591 47.6507 **262** 51.3826 47.7779 **278** 51.0105 47.9036 **292** 49.8925 48.2812 **252** 47.3681 48.1956 **232** 44.2257 48.7578 **269** 43.3617 48.9124 **312** 43.3323 48.9315 **286** 43.0534 52.1861 **179** 41.9086

52.747 **21** 40.4046 53.7748 **197** 40.4492 55.5288 **174** 40.3607 56.4453 **54** 38.9841 57.2395 **99** 38.3772 57.5896 **277** 36.8609 55.6675 **86** 34.3161 52.4409 **270** 34.0589 52.4955 **228** 33.1907 52.6797 **172** 32.5782 53.0831 **196** 31.2055 53.987 **287** 30.7719 54.2726 **60** 29.9047 55.4454 **215** 30.314 57.5158 **259** 29.4242 57.6896 **74** 28.3741 57.8947 **238** 26.5886 58.2434 **280** 24.967 58.4483 **206** 23.9024 62.8167 **137** 22.2443 64.4163 **290** 20.2121 66.3767 **258** 18.6233 66.3764 **1** 17.2058 66.376 **300** 12.6416 66.3757 **159** 8.87721 66.3756 **94** 2.15979 67.5328 **142** 4.55442 61.8516 **11** 4.99307 61.0406 **276** 7.88605 55.6923 **307** 8.75485 54.0862 **76** 9.62364 52.4802 **191** 10.1555 51.497 **187** 8.50343 46.6013 **162** 6.08841 39.4447 **163** 15.5233 41.5638 **209** 17.0913 42.5499 **248** 17.2415 43.4383 **239** 16.5746 44.4504 **26** 15.6776 45.8116 **73** 15.5457 46.0118 **309** 18.0529 49.6637 **89** 19.4436 49.3476 **260** 20.9676 49.0012 **267** 25.7535 46.7446 **261** 27.1891 46.0676 **189** 28.0799 45.6476 **318** 29.03 45.1996 **317** 29.6206 42.629 **245** 30.9208 36.9695 **203** 30.9208 36.9695 **34** 31.2177 37.4342 **200** 32.3413 39.1929 **180** 33.1811 40.3114 **152** 36.0345 44.1102 **275** 36.1923 44.0101 **194** 36.9548 43.5264 **324** 37.7245 38.8967 **303** 39.5627 37.823 **78** 42.5194 36.0962 **301** 44.3852 39.229 **217** 44.6675 39.2825 **57** 44.6792 39.2847 **272** 47.6139 38.059 **165** 50.3556 34.8348 **255** 50.8251 34.334 **84** 52.3065 32.7539 **153** 52.3841 32.6712 **246** 52.8826 32.1397 **183** 54.9419 37.2696 **314** 54.2712 38.6799 **302** 56.3579 41.542 **59** 56.9242 40.8862 **87** 57.953 39.6946 **284** 59.4843 36.5197 **105** 59.5541 36.3661 **75** 59.5971 36.2712 **122** 61.5672 31.9324 **251** 65.4343 28.9838 **192** 65.6078 28.4687 **298** 65.8496 21.5791 **218** 65.8826 20.6396 **315** 65.9408 18.9877 **225** 66.0116 16.9794 **145** 60.836 19.4303 **198** 56.9975 21.249 **240** 56.0648 23.8135 **293** 55.6599 24.2405 **308** 53.0385 24.1549 **325** 50.7879 23.7433 **81** 48.0471 21.9003 **119** 42.1233 26.597 **136** 38.9422 26.5986 **18** 34.1186 26.601 **32** 28.232 28.6582 **141** 27.5592 30.5651 **247** 26.5034 33.5558 **299** 25.3863 33.2741 **205** 22.8858 32.6434 **100** 19.7302 34.2398 **27** 19.512 32.4256 **90** 19.4012 27.7645 **147** 16.5255 26.2631 **15** 16.4637 26.1035 **296** 18.1695 23.4865 **289** 18.984 22.2369 **148** 19.0277 22.1699 **51** 19.0629 22.1159 **328** 12.7332 15.469 **181** 12.5423 15.3711 **103** 6.2542 18.5173 **133** 1.58154 20.8552 **10** 1.97589 13.8971

Funding Open access funding provided by the Scientific and Technological Research Council of Türkiye (TÜBİTAK).

Data availability Data can be made available by the corresponding author upon request.

Declarations

Conflict of interest There is no Conflict of interest between authors.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Lin S (1965) Computer solutions of the traveling salesman problem. Bell Syst Tech J 44(10):2245–2269
2. Gutin G, Punnen A P (Eds.) (2006). The traveling salesman problem and its variations (Vol. 12). Springer Science & Business Media
3. Di Placido A, Archetti C, Cerrone C, Golden B (2023) The generalized close enough traveling salesman problem. Eur J Oper Res 310(3):974–991
4. Lee Y W, Eun S, Oh S H (2008). Wireless digital water meter with low power consumption for automatic meter reading. In 2008 International Conference on Convergence and Hybrid Information Technology (pp. 639-645). IEEE

5. Di Placido A, Archetti C, Cerrone C (2022) A genetic algorithm for the close-enough traveling salesman problem with application to solar panels diagnostic reconnaissance. Comput Oper Res 145:105831
6. Jan GE, Chang KY, Parberry I (2008) Optimal path planning for mobile robot navigation. IEEE/ASME Trans Mechatron 13(4):451–460
7. Evers L, Dollevoet T, Barros AI, Monsuur H (2014) Robust UAV mission planning. Ann Oper Res 222:293–315
8. Caraffini F, Neri F, Iacca G, Mol A (2013) Parallel memetic structures. Inf Sci 227:60–82
9. Alba E (2005) Parallel metaheuristics: a new class of algorithms. John Wiley & Sons
10. Neri F, Cotta C (2012) Memetic algorithms and memetic computing optimization: A literature review. Swarm Evol Comput 2:1–14
11. Moscato P, Cotta C, Mendes A (2004) Memetic algorithms. New Optim Tech Eng 141:53–85
12. Lei Z, Hao JK (2024) An effective memetic algorithm for the close-enough traveling salesman problem. Appl Soft Comput 153:111266
13. Chandra R (2001) Parallel programming in OpenMP. Morgan Kaufmann
14. Ayguadé E, Copty N, Duran A, Hoeflinger J, Lin Y, Massaioli F, Zhang G (2008) The design of OpenMP tasks. IEEE Trans Parallel Distrib Syst 20(3):404–418
15. Dülger Ö, Dökeroglu T (2024) A new parallel tabu search algorithm for the optimization of the maximum vertex weight clique problem. Concurr Comput Pract Exper 36(2):e7891
16. Gulczynski D J, Heath J W, Price C C (2006). The close enough traveling salesman problem: A discussion of several heuristics. Perspectives in Operations Research: Papers in Honor of Saul Gass' 80 th Birthday, 271-283
17. Carrabs F, Cerrone C, Cerulli R, Gaudioso M (2017) A novel discretization scheme for the close enough traveling salesman problem. Comput Oper Res 78:163–171
18. Deckerová J, Váňa P, Faigl J (2024) Combinatorial lower bounds for the generalized traveling salesman problem with neighborhoods. Expert Syst Appl 258:125185
19. Katoch S, Chauhan SS, Kumar V (2021) A review on genetic algorithm: past, present, and future. Multimedia Tools Appl 80:8091–8126
20. Coutinho WP, Nascimento RQD, Pessoa AA, Subramanian A (2016) A branch-and-bound algorithm for the close-enough traveling salesman problem. INFORMS J Comput 28(4):752–765
21. Behdani B, Smith JC (2014) An integer-programming-based approach to the close-enough traveling salesman problem. INFORMS J Comput 26(3):415–432
22. Carrabs F, Cerrone C, Cerulli R, Golden B (2020) An adaptive heuristic approach to compute upper and lower bounds for the close-enough traveling salesman problem. INFORMS J Comput 32(4):1030–1048
23. Deckerová J, Kučerová K, Faigl J (2023). On improvement heuristic to solutions of the close enough traveling salesman problem in environments with obstacles. In 2023 European Conference on Mobile Robots (ECMR) (pp. 1-6). IEEE
24. Kennedy J, Eberhart R (1995, November). Particle swarm optimization. In Proceedings of ICNN'95-international conference on neural networks (Vol. 4, pp. 1942-1948). ieee
25. Dokeroglu T, Sevinc E, Kucukyilmaz T, Cosar A (2019) A survey on new generation metaheuristic algorithms. Comput Indus Eng 137:106040
26. Yang Z, Xiao MQ, Ge YW, Feng DL, Zhang L, Song HF, Tang XL (2018) A double-loop hybrid algorithm for the traveling salesman problem with arbitrary neighbourhoods. Eur J Oper Res 265(1):65–80
27. Cariou C, Moiroux-Arvis L, Pinet F, Chanet JP (2023) Evolutionary algorithm with geometrical heuristics for solving the Close Enough Traveling Salesman Problem: Application to the trajectory planning of an Unmanned Aerial Vehicle. Algorithms 16(1):44
28. Faigl J (2018) GSOA: growing self-organizing array-unsupervised learning for the close-enough traveling salesman problem and other routing problems. Neurocomputing 312:120–134
29. Carrabs F, Cerulli R, D'Ambrosio C, Murano G (2025). Upper Bound Computation for the Multiple Close-Enough Traveling Salesman Problem. In Proceedings of the 14th International Conference on Operations Research and Enterprise Systems-ICORES (pp. 186-195)
30. Toman O (2025). Toward optimal solution of the close enough traveling salesman problem using interval sampling. Master's thesis
31. Dokeroglu T, Canturk D (2025). New Harris Hawks algorithms for the Close-Enough Traveling Salesman Problem. Intelligent Systems with Applications, 200586
32. Wang X, Golden B, Wasil E (2019) A steiner zone variable neighborhood search heuristic for the close-enough traveling salesman problem. Comput Oper Res 101:200–219
33. Mennell WK (2009) Heuristics for solving three routing problems: Close-enough traveling salesman problem, close-enough vehicle routing problem, and sequence-dependent team orienteering problem. University of Maryland, College Park
34. Padilha G A G, Jung J J, de Mattos Neto P S (2024). Memetic algorithm-based optimization of hybrid forecasting systems for multivariate time series. Neural Computing and Applications, 1-18

35. Hansen P, Mladenović N, Todosijević R, Hanafi S (2017) Variable neighborhood search: basics and variants. EURO J Comput Optim 5(3):423–454
36. Helsgaun K (2000) An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur J Oper Res 126(1):106–130
37. Aslan Y, Candotti M, Yarovoy A (2019). Synthesis of multi-beam space-tapered linear arrays with side lobe level minimization in the presence of mutual coupling. In 2019 13th European Conference on Antennas and Propagation (EuCAP) (pp. 1-5). IEEE
38. Lei Z, Hao JK (2023) A memetic algorithm for the close-enough traveling salesman problem. In EU/Meeting, Troyes, France
39. Tang J, Lim MH, Ong YS (2007) Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems. Soft Comput 11:873–888
40. Nalepa J, Blocho M (2015) Co-operation in the parallel memetic algorithm. Int J Parallel Prog 43:812–839
41. Sörensen K, Sevaux M (2006) MA|PM: memetic algorithms with population management. Comput Oper Res 33(5):1214–1225
42. D'Angelo G, Palmieri F (2021) GGA: A modified genetic algorithm with gradient-based local search for solving constrained optimization problems. Inf Sci 547:136–162
43. Alhijawi B, Awajan A (2024) Genetic algorithms: Theory, genetic operators, solutions, and applications. Evol Intel 17(3):1245–1256
44. Gurobi Optimization, LLC, (2023) Gurobi Optimizer Reference Manual, https://www.gurobi.com
45. López-Ibáñez M, Dubois-Lacoste J, Cáceres LP, Birattari M, Stützle T (2016) The irace package: Iterated racing for automatic algorithm configuration. Oper Res Perspect 3:43–58
46. Bixby B, Reinelt G (1991) TSPLIB, A traveling salesman problem library. ORSA J Comput 3:376–384
47. Chao IM (1993) Algorithms and solutions to multi-level vehicle routing problems. University of Maryland, College Park

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Deniz Canturk[1] · Tansel Dökeroğlu[1]

✉ Deniz Canturk
deniz.canturk@tedu.edu.tr

[1] Department of Software Engineering, TED University, Ankara, Turkey