

META-HEURISTIC ALGORITHMS FOR QUASI TOTAL DOUBLE ROMAN DOMINATION PROBLEM

CHARAN KARNATI¹, ALFRED RAJU M² AND PALAGIRI VENKATA SUBBA REDDY^{3,*}

Abstract. Ensuring the resilience and security of complex networks, such as communication or power grids, requires strategies that can withstand failures and attacks. One such approach involves the use of domination models in graph theory. In this work, we focus on the quasi total double Roman domination problem (QTDRDP), a combinatorial optimization problem that models multi-layered defense strategies on graphs. Solving this problem involves assigning integer weights to nodes under specific constraints, aiming to minimize the total weight while maintaining strong structural security. Since computing the optimal assignment is NP-hard, we propose two metaheuristic approaches: one based on genetic algorithm (GA) and the other on artificial bee colony (ABC) optimization. To evaluate their effectiveness, we conducted experiments across multiple graph instances. The ABC algorithm achieved 120 successes out of 124 trials, significantly outperforming the GA algorithm. Statistical analysis confirms this difference is highly significant, with a critical p-value of $4.560507128 \times 10^{-31}$, leading us to reject the null hypothesis. These results demonstrate that the ABC approach is more robust and efficient for solving the QTDRDP.

Mathematics Subject Classification. 05C69, 05C85, 90C27, 68W50.

Received March 17, 2025. Accepted August 1, 2025.

1. INTRODUCTION

In computer networking, monitoring nodes are crucial for effective network management and timely issue detection. However, due to budget limitations, these nodes can only be installed in selected network segments. We represent the network as a simple, undirected graph $G(V, E)$, where each vertex represents a network segment and each edge a connection between segments. A network segment is considered *secured* if it either has a monitoring node itself or is adjacent to at least two segments that have monitoring nodes. Placing a monitoring node at every segment is both costly and inefficient, making this problem one of optimal resource allocation.

Keywords and phrases: Quasi total double Roman domination, NP-Hard, Roman domination, artificial bee colony algorithm, genetic algorithm.

¹ Department of Computer Science and Engineering, National Institute of Technology Andhra Pradesh, Tadepalligudem 534101, Andhra Pradesh, India.

² Department of Computer Science and Engineering, National Institute of Technology Warangal, Hanamkonda 506004, Hanamkonda, Telangana, India.

³ Department of Computer Science and Engineering, National Institute of Technology Warangal, Hanamkonda 506004, Telangana, India.

* Corresponding author: pvsr@nitw.ac.in

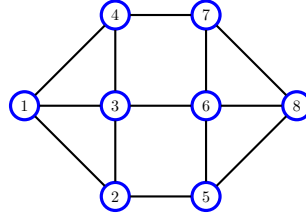


FIGURE 1. A simple and undirected graph.

To model this, we use the concept of *domination* in graphs. Let $V(G)$ stand for the set of vertices and $E(G)$ for the set of edges of graph G . The *open neighborhood* of a vertex v is represented by $N(v) = \{u \mid (u, v) \in E\}$ and the *closed neighborhood* $N[v]$ of a vertex v is represented by $N(v) \cup \{v\}$. A dominating set in a graph is a subset of vertices $D \subseteq V$ such that every vertex in $V \setminus D$ is adjacent to at least one vertex in D . Determining the minimum number of nodes to accomplish this task leads to the concept of graph theory known as **domination**, central to optimizing resource allocation in applications like network monitoring, where budget constraints limit the number of monitoring nodes [1, 2].

To illustrate various domination functions in graph theory, we consider a simple undirected graph depicted in Figure 1, consisting of 8 vertices and 13 edges. We define two labeling functions:

- h : a correct labeling function that satisfies the respective domination constraints.
- g : an incorrect labeling function that violates some constraint, with explanations provided.
- **Roman Domination** — A Roman dominating function (RDF) is a function $h : V \rightarrow \{0, 1, 2\}$ such that every vertex $v \in V$ with $h(v) = 0$ has at least one neighbor $u \in V$ with $h(u) = 2$ [3].
RDF labeling (valid):

$$h = \{1 : 2, 2 : 0, 3 : 0, 4 : 0, 5 : 0, 6 : 0, 7 : 0, 8 : 2\}$$

RDF labeling (invalid):

$$g = \{1 : 0, 2 : 1, 3 : 1, 4 : 0, 5 : 0, 6 : 2, 7 : 1, 8 : 0\}$$

Violations: Vertex 1 is labeled 0, and in its neighbors, no neighbor has label 2. Vertex 4 is labeled 0, and neighbors (1, 3, 7) have labels (0, 1, 1) — again, no neighbor with label 2.

- **Double Roman Domination** — A function $h : V(G) \rightarrow \{0, 1, 2, 3\}$ is a *Double Roman Dominating Function* if it satisfies:

1. For any vertex v with $h(v) = 0$, there exists either:
 - at least one neighbor u of v such that $h(u) = 3$, or
 - at least two neighbors u and w of v such that $h(u) = h(w) = 2$.
2. For any vertex v with $h(v) = 1$, there exists at least one neighbor u of v such that $h(u) \geq 2$.

DRDF labeling (valid):

$$h = \{1 : 0, 2 : 2, 3 : 3, 4 : 0, 5 : 0, 6 : 3, 7 : 1, 8 : 0\}$$

DRDF labeling (invalid):

$$g = \{1 : 0, 2 : 2, 3 : 2, 4 : 0, 5 : 0, 6 : 2, 7 : 1, 8 : 0\}$$

Violations: Vertex 4 has only one neighbor (3) labeled 2 — not enough for DRDF. Vertex 8 has only one neighbor (6) labeled 2 — also violates DRDF.

- **Total Double Roman Domination** — A total double Roman dominating function (TDRDF) is a DRDF such that the subgraph induced by the non-zero labeled vertices has no isolated nodes [4].

TDRDF labeling (valid):

$$h = \{1 : 1, 2 : 2, 3 : 2, 4 : 1, 5 : 0, 6 : 3, 7 : 2, 8 : 1\}$$

TDRDF labeling (invalid):

$$g = \{1 : 0, 2 : 2, 3 : 0, 4 : 3, 5 : 0, 6 : 2, 7 : 0, 8 : 2\}$$

This labeling is a valid DRDF but an invalid TDRDF because vertices 2 and 4 do not follow the condition *i.e.* the subgraph induced by non-zero labeled vertices has no isolated nodes.

- **Quasi Total Double Roman Domination** — This variant [5] imposes:
 - If $f(v) = 0$, then v must have one neighbor with label 3 or at least two neighbors with label 2.
 - If $f(v) = 1$, then v must have at least one neighbor with label ≥ 2 .
 - Any isolated vertex in the subgraph of non-zero labeled vertices must be labeled 2.

QTDRDF labeling (valid):

$$h = \{1 : 0, 2 : 2, 3 : 0, 4 : 3, 5 : 0, 6 : 2, 7 : 1, 8 : 2\}$$

QTDRDF labeling (invalid):

$$g = \{1 : 0, 2 : 2, 3 : 0, 4 : 3, 5 : 0, 6 : 2, 7 : 0, 8 : 1\}$$

Violation: Vertex 4 is labeled 3 and does not have supporting label 1 or label 2, which disobeys the third condition in the definition of QTDRDF.

Finding a solution to the quasi total double Roman domination problem allows it to be used in a variety of disciplines where resource allocation is required, *i.e.*, cellular tower installation, fire station installation, *etc.* As there is no deterministic strategy for solving this problem in polynomial time, the emphasis is on metaheuristics, which, although not necessarily ensuring an optimal solution, can produce near-optimal solutions to difficult problems in a limited amount of time or repetitions.

The rest of the paper is structured as follows. Literature review, heuristics, genetic algorithm and the artificial bee colony algorithm based solutions followed by experimental results and conclusion.

2. LITERATURE REVIEW

The optimization version of the quasi-total double Roman domination problem is described as follows:

Minimum Quasi-Total Double Roman Domination Problem (MQTDRDP)

Input: A graph $G(V, E)$

Output: $\gamma_{qtdR}(G)$

The quasi-total double Roman domination function was first studied by S. Kosari *et al.* in 2024 [5]. Kosari's work demonstrated that MQTDRDP is NP-complete, even for bipartite graphs, and provided an upper bound for this domination number. It has also been proved that for path graphs, P_n (where $n \geq 2$): $\gamma_{qtdR}(P_n) = n + 1$ and for cycle graphs C_n (where $n \geq 3$): $\gamma_{qtdR}(C_n) = n + (n \bmod 2)$. The inequality $\gamma_{dR}(G) \leq \gamma_{qtdR}(G) \leq \gamma_{tdR}(G)$ was established, where $\gamma_{dR}(G)$ is the double Roman domination number and $\gamma_{tdR}(G)$ is the total double Roman domination number of graph G .

In 2020, Khandelwal *et al.* proposed a genetic algorithm-based solution for the Roman domination problem [6]. Following this, Jakob Greilhuber introduced a simulated annealing-based approach for Roman domination problem in 2023 [7]. The concept of Roman domination has also been effectively applied to neural network problems in the healthcare sector, highlighting its broad relevance and utility across diverse domains [8]. Heuristic methods for solving domination problems were earlier presented by Chaurasia in 2015 [9], and metaheuristic algorithms for the double Roman domination problem were discussed by Aggarwal *et al.* in 2024 [10]. Recently, M. Alfred Raju *et al.* proposed metaheuristic algorithms for the Roman{2}- domination problem [11].

Despite these advancements, no metaheuristic based approach has yet been discovered for determining the quasi-total double Roman domination number of a graph. In this research, two solutions are proposed for solving MQTDRDP and evaluated on various datasets using genetic algorithm and an artificial bee colony (ABC) optimization approach.

The selection of Genetic Algorithm (GA) and Artificial Bee Colony (ABC) for solving the Quasi Total Double Roman Domination Problem (QTDRDP) is justified by their suitability for combinatorial and NP-hard problems, with GA's crossover and mutation effectively exploring large solution spaces and ABC's swarm-based foraging balancing exploration and exploitation to avoid local optima. Compared to alternatives like Particle Swarm Optimization (less effective for discrete problems), Simulated Annealing (requiring complex cooling schedules), Ant Colony Optimization (better for path-based problems), or Tabu Search (needing intricate memory management), GA and ABC offer structured exploration and adaptability. Preliminary testing showed ABC outperforming GA (120 successes vs. 4 failures), likely due to its employed-onlooker-scout mechanism, while GA's widespread use in similar domination problems (*e.g.*, Roman Domination) made it a reliable baseline. Literature precedents and simpler implementation further support their choice over less-tested or complex metaheuristics like Differential Evolution or Harmony Search.

3. GENETIC ALGORITHM APPROACH FOR QTDRDP

A genetic algorithm (GA) is an optimization technique inspired by natural evolution and genetics, particularly effective for solving complex, NP-hard problems like the Quasi Total Double Roman Domination Problem (QTDRDP), where traditional methods may be inefficient [12, 13]. Solutions are represented as chromosomes, typically encoded as vectors of genes. The algorithm begins with an initial population of chromosomes, generated either randomly or heuristically. Through iterative generations, it selects fitter chromosomes (those with better objective values) for reproduction, applies crossover to combine parent chromosomes to produce offspring, and introduces mutation to maintain diversity and explore new solution regions. The fitness of each chromosome is evaluated, and the process continues until a termination condition, such as a maximum number of generations or an acceptable fitness level, is met. The key operations include:

- **Initialization:** Generate an initial population of chromosomes.
- **Selection:** Choose chromosomes for reproduction based on fitness, using methods like tournament selection.
- **Crossover:** Combine pairs of parent chromosomes to produce offspring, using operators like two-point crossover.
- **Mutation:** Apply random changes to chromosomes to ensure diversity.
- **Fitness Evaluation:** Compute the fitness of each chromosome to guide selection.
- **Termination:** Continue until a stopping criterion is satisfied.

The GA for QTDRDP, detailed in Algorithm 1, uses these operations to minimize the quasi total double Roman domination number $\gamma_{qtdR}(G)$, leveraging its ability to handle combinatorial optimization in graph problems [14].

3.1. Individual representation

In the proposed GA, a chromosome represents a candidate solution to the QTDRDP. For a graph $G = (V, E)$ with vertex set $V = \{v_1, v_2, \dots, v_n\}$, a chromosome is encoded as a vector $S = [S[v_1], S[v_2], \dots, S[v_n]]$, where

$S[v_i] \in \{0, 1, 2, 3\}$ denotes the label assigned to vertex v_i . This corresponds to a quasi total double Roman dominating function $f : V \rightarrow \{0, 1, 2, 3\}$, satisfying:

- If $S[v_i] = 0$, then v_i has at least two neighbors with label 2 or one neighbor with label 3.
- If $S[v_i] = 1$, then v_i has at least one neighbor with label ≥ 2 .
- If v_i is an isolated vertex in the subgraph induced by vertices with non-zero labels ($\{v \mid S[v] \geq 1\}$), then $S[v_i] = 2$.

The initial population is generated using Heuristics 1, 2, and 3 (Algorithms 6, 7, 8), and also with 65% from Heuristic 3, 20% from Heuristic 1, and 15% from Heuristic 2, ensuring diversity and feasible or near-feasible solutions. This vector-based representation is standard for graph labeling problems, such as Roman domination [6] and double Roman domination [10], and is compatible with GA operations.

3.2. Fitness function

The fitness function evaluates the chromosome $S = [S[v_1], S[v_2], \dots, S[v_n]]$ by computing the total weight of the quasi total double Roman dominating function:

$$\text{Fitness}(S) = \sum_{v_i \in V} S[v_i]$$

The objective is to minimize this fitness score. Chromosomes with lower fitness scores are fitter, as they represent solutions with fewer resources (lower labels) while satisfying the domination constraints. The feasibility function (Algorithm 2) ensures that only valid solutions are evaluated, preventing infeasible chromosomes from skewing the optimization process [13].

3.3. Selection and crossover operator

We have used the tournament selection method to choose parent chromosomes for reproduction [14]. In each tournament, a subset of chromosomes of specific size is randomly selected, and one is chosen as a parent, allowing less fit individuals a chance to contribute to diversity. This process is repeated to select two parents. The two-point crossover operator then combines these parents to produce offspring. Two random crossover points are chosen within the chromosome length, and the segments between these points are swapped between parents. For example, if parents P_1 and P_2 have crossover points at indices 3 and 7, the genes from indices 3 to 6 are exchanged, producing two offspring.

The two-point crossover is chosen for its simplicity and ability to combine diverse segments of parent solutions, promoting exploration of the solution space. However, as it is not tailored to preserve QTDRDP constraints, it may produce infeasible offspring, necessitating the feasibility function (Algorithm 2). This trade-off between operator simplicity and solution repair is justified by the NP-hard nature of QTDRDP, where constraint-preserving crossovers would require significant computational overhead.

3.4. Feasibility function and mutation operator

Due to the complexity of QTDRDP constraints, the two-point crossover and mutation operators may produce offspring that violate the domination requirements. For instance, swapping segments during crossover can result in a vertex with label 0 lacking sufficient neighbors with labels 2 or 3, or a vertex with label 1 having no neighbors with label ≥ 2 . The mutation operator, which randomly changes a vertex's label to a value in $\{0, 1, 2, 3\}$ with a small probability (*e.g.*, 0.01), may similarly disrupt feasibility [12].

The feasibility function (Algorithm 2) ensures that all chromosomes represent valid QTDRD functions by checking and repairing each vertex's label based on its neighbors:

Algorithm 1 Genetic algorithm approach

```

1: Input: Graph  $G$ ,  $population\_size$ ,  $generations$ ,  $tournament\_size$ 
2: Output: Final  $population$ 
3: Initialize  $population$  with  $population\_size$  heuristically generated solutions
4: while  $gen \leq generations$  do
5:   while  $|population| < population\_size$  do
6:      $parent1 \leftarrow \text{TournamentSelection}(population, tournament\_size)$ 
7:      $parent2 \leftarrow \text{TournamentSelection}(population, tournament\_size)$ 
8:      $offspring1, offspring2 \leftarrow \text{Crossover}(parent1, parent2)$ 
9:     if  $\text{FeasibilityCheck}(offspring1, G)$  then
10:      Add  $offspring1$  to  $population$ 
11:     end if
12:     if  $\text{FeasibilityCheck}(offspring2, G)$  then
13:      Add  $offspring2$  to  $population$ 
14:     end if
15:   end while
16:   Sort  $population$  by fitness
17:    $population \leftarrow$  First  $population\_size$  solutions from  $population$ 
18:    $gen \leftarrow gen + 1$ 
19: end while
20: return  $population$ 

```

- If $S[v_i] = 0$, it verifies whether v_i has at least two neighbors with label 2 or one neighbor with label 3. If not, it adjusts $S[v_i]$ to 1 (if one neighbor has label 2 or a label-3 neighbor has positive-labeled neighbors) or 2 (if no neighbors have label 2).
- If $S[v_i] = 1$, it ensures v_i has at least one neighbor with label ≥ 2 . If not, $S[v_i]$ is set to 2.
- It checks the subgraph induced by vertices with non-zero labels to ensure isolated vertices have label 2.

This repair mechanism allows the GA to use standard crossover and mutation operators while maintaining solution validity. The use of a feasibility function is justified by the NP-hard nature of QTDRDP, where designing operators that always produce feasible solutions is computationally expensive.

4. ARTIFICIAL BEE COLONY ALGORITHM FOR QTDRDP

The Artificial Bee Colony (ABC) algorithm, introduced by Karaboga in 2005 [15], is a nature-inspired optimization technique based on the foraging behavior of honey bees. It balances exploration and exploitation to solve complex combinatorial problems like the Quasi total double Roman domination problem (QTDRDP). The algorithm models a population of food sources (solutions) manipulated by three types of bees: employed bees, onlooker bees, and scout bees. Employed bees explore new solutions near their assigned food sources, onlooker bees select food sources based on their fitness (quality), and scout bees replace stagnant food sources with new ones to maintain diversity. The ABC algorithm operates in three phases—employed bee, onlooker bee, and scout bee—described below, along with its implementation details for QTDRDP. The algorithm has been widely studied and applied to various optimization problems [15–17].

4.1. Individual representation

In the ABC algorithm, a food source represents a candidate solution to the QTDRDP. For a graph $G = (V, E)$ with $|V| = n$, a food source is encoded as a vector $S = [S[v_1], S[v_2], \dots, S[v_n]]$, where $S[v_i] \in \{0, 1, 2, 3\}$ is the label assigned to vertex v_i . This vector corresponds to a quasi total double Roman dominating function.

Algorithm 2 Feasibility Function

```

1: Input: Solution, Graph G
2: Output: Updated feasible solution
3: Function Feasibility(solution, G)
4: for  $i = 0$  to length(solution) - 1 do
5:     neighbors  $\leftarrow$  labels of G[i] in solution
6:     if solution[i] = 0 then
7:         if  $3 \in$  neighbors and Not All neighbors of label-3 neighbors of i are positive then
8:             solution[i]  $\leftarrow$  1
9:         else if count of 2 in neighbors == 1 then
10:            solution[i]  $\leftarrow$  1
11:        else if  $2 \notin$  neighbors then
12:            solution[i]  $\leftarrow$  2
13:        end if
14:    else if solution[i] = 1 and  $2 \notin$  neighbors then
15:        solution[i]  $\leftarrow$  2
16:    end if
17: end for
18: return solution
    
```

The initial population of food sources is generated using 65% from Heuristic 3, 20% from Heuristic 1, and 15% from Heuristic 2, ensuring diversity and feasibility. This vector-based representation is standard for graph labeling problems, such as Roman domination [6] and double Roman domination [10].

4.2. Fitness function

The fitness function evaluates the quality of a food source $S = [S[v_1], S[v_2], \dots, S[v_n]]$ by computing the total weight of the quasi total double Roman dominating function:

$$\text{Fitness}(S) = \sum_{v_i \in V} S[v_i]$$

The goal is to minimize this fitness score, which corresponds to the quasi total double Roman domination number $\gamma_{qtdR}(G)$. Lower fitness scores indicate better solutions. Feasibility is enforced through a dedicated function (described below), ensuring that only valid solutions are evaluated.

4.3. Feasibility function

The ABC algorithm uses a feasibility function (applied in Algorithms 3, 4, and 5) to ensure that all food sources represent valid QTDRD functions. The neighborhood exploration formula $v_{ij} = x_{ij} + \phi_{ij} \times (x_{ij} - x_{kj})$ may produce non-integer or out-of-range labels, which are clipped to $\{0, 1, 2, 3\}$. However, clipped solutions may still violate QTDRDP constraints, such as a vertex with label 0 lacking sufficient neighbors with labels 2 or 3.

The feasibility function repairs such solutions by checking each vertex's label against its neighbors' labels:

- For a vertex v_i with $S[v_i] = 0$, it ensures at least two neighbors have label 2 or one neighbor has label 3. If not, it adjusts $S[v_i]$ to 1 or 2 based on neighbor labels.
- For a vertex v_i with $S[v_i] = 1$, it verifies the presence of at least one neighbor with label ≥ 2 . If absent, $S[v_i]$ is set to 2.
- It ensures that any isolated vertex in the subgraph induced by non-zero labels has label 2.

Algorithm 3 Employed Bee Phase

```

1: Input: Food sources,  $n$  (number of elements in food source)
2: Output: Updated Food sources
3: for  $i \leftarrow 0$  to NUM_EMP_BEES  $- 1$  do
4:    $fi \leftarrow i$ 
5:    $k \leftarrow$  random different index
6:   for  $j \leftarrow 0$  to  $n - 1$  do
7:      $\phi \leftarrow$  random double
8:     newSol[ $j$ ]  $\leftarrow$  food[ $fi$ ].sol[ $j$ ] +  $\phi \times$  (food[ $fi$ ].sol[ $j$ ] - food[ $k$ ].sol[ $j$ ])
9:     newSol[ $j$ ]  $\leftarrow$  clip(newSol[ $j$ ], 0, 3)
10:  end for
11:  if isFeasible(newSol, adjList) then
12:    newFit  $\leftarrow$  computeFitness(newSol)
13:    if newFit < food[ $fi$ ].fit then
14:      food[ $fi$ ].sol  $\leftarrow$  newSol
15:      food[ $fi$ ].fit  $\leftarrow$  newFit
16:      food[ $fi$ ].trials  $\leftarrow$  0
17:    else
18:      food[ $fi$ ].trials  $\leftarrow$  food[ $fi$ ].trials + 1
19:    end if
20:  else
21:    food[ $fi$ ].trials  $\leftarrow$  food[ $fi$ ].trials + 1
22:  end if
23: end for

```

This repair mechanism, detailed in Algorithm 2 for the genetic algorithm, is similarly applied here. The need for a feasibility function arises from the generic nature of the neighborhood exploration, which is not tailored to preserve QTDRDP constraints. While constraint-preserving operators could reduce repairs, they would increase complexity. The current approach balances exploration and feasibility effectively for this NP-hard problem.

4.4. Employed bee phase

In the employed bee phase, each employed bee explores a new solution near its assigned food source. A new candidate solution v_{ij} is generated for the i -th food source using:

$$v_{ij} = x_{ij} + \phi_{ij} \times (x_{ij} - x_{kj})$$

where x_{ij} is the current solution's j -th component, $\phi_{ij} \in [-1, 1]$ is a random number, x_{kj} is a randomly chosen solution's j -th component, and j is a randomly selected dimension. The new solution is clipped to $\{0, 1, 2, 3\}$ and checked for feasibility. If feasible and its fitness (sum of labels) is lower than the current solution's, it replaces the current solution, and the trial counter is reset; otherwise, the trial counter increments (Algorithm 3). Ten employed bees are deployed over 1000 food sources.

4.5. Onlooker bee phase

In the onlooker bee phase, onlooker bees select food sources based on their probabilities, computed as:

$$p_i = \frac{f_i}{\sum_{j=1}^N f_j}$$

Algorithm 4 Onlooker Bee Phase

```

1: Input: Food sources, probabilities
2: Output: Updated Food sources
3: for  $t = 1$  to NUM_ONLOOKER_BEES do
4:    $i \leftarrow \text{SelectFoodSource}(\text{probabilities})$ 
5:    $k \leftarrow \text{RandomFoodSourceExcept}(i)$ 
6:    $j \leftarrow \text{rand}(0, n - 1)$ 
7:    $\phi \leftarrow \text{rand}(-1, 1)$ 
8:    $\text{new\_solution} \leftarrow \text{food}[i].\text{solution}$ 
9:    $\text{new\_solution}[j] \leftarrow \text{clamp}(\text{new\_solution}[j] + \phi \times (\text{food}[i].\text{solution}[j] - \text{food}[k].\text{solution}[j]), 0, 3)$ 
10:   $\text{new\_solution} \leftarrow \text{apply\_feasibility}(\text{new\_solution}, \text{adj\_list})$ 
11:   $\text{new\_fitness} \leftarrow \text{evaluate\_objective\_function}(\text{new\_solution})$ 
12:  if  $\text{new\_fitness} < \text{food}[i].\text{fitness}$  then
13:     $\text{food}[i] \leftarrow \{\text{new\_solution}, \text{new\_fitness}, 0\}$ 
14:  else
15:     $\text{food}[i].\text{trial} \leftarrow \text{food}[i].\text{trial} + 1$ 
16:  end if
17: end for

```

Algorithm 5 Scout Bee Phase

```

1: Input: Food sources, adj_list
2: Output: Updated Food sources
3: for each  $\text{food}$  in  $\text{food\_sources}$  do
4:   if  $\text{food.trial} > \text{LIMIT}$  then
5:      $\text{food.solution} \leftarrow \text{heuristic}(\text{adj\_list})$ 
6:      $\text{food.solution} \leftarrow \text{apply\_feasibility}(\text{food.solution}, \text{adj\_list})$ 
7:      $\text{food.fitness} \leftarrow \text{evaluate\_fitness}(\text{food.solution})$ 
8:      $\text{food.trial} \leftarrow 0$ 
9:   end if
10: end for

```

where f_i is the fitness value of the i -th food source, and N is the number of food sources. A new solution is generated using the same formula as in the employed bee phase, clipped, and checked for feasibility. If the new solution's fitness is lower, it replaces the current solution; otherwise, the trial counter increments (Algorithm 4). Ten onlooker bees are deployed over 1000 food sources, continuing until all onlooker bees have made their choices.

4.6. Scout bee phase

In the scout bee phase, food sources that have not improved after a predefined trial limit are abandoned. The associated employed bee becomes a scout bee and reinitializes the food source using one of the three heuristics (Algorithms 6, 7, 8). The new solution is checked for feasibility, its fitness is computed, and the trial counter is reset (Algorithm 5). This phase prevents stagnation by introducing new solutions, maintaining diversity in the population.

4.7. Implementation details

The ABC algorithm is implemented with a population of 1000 food sources, initialized using a combination of Heuristics 1, 2, and 3 (65% Heuristic 3, 20% Heuristic 1, and 15% Heuristic 2). Ten employed bees and ten onlooker bees are used, balancing exploration and exploitation. The trial limit for abandoning a food source is

Algorithm 6 Heuristic 1

```

1: Input: A simple and undirected graph  $G = (V, E)$ 
2: Output: Feasible solution  $S$ 
3:  $V' \leftarrow V$ 
4: while  $V'$  is not empty do
5:    $u_1 \leftarrow$  Random vertex from  $V'$ 
6:    $u_2 \leftarrow$  Random vertex from  $V'$  such that  $u_2 \neq u_1$ 
7:    $S[u_1, u_2] \leftarrow 2$ 
8:   for each vertex  $w \in (N(u_1) \cup N(u_2)) \setminus \{u_1, u_2\}$  do
9:     if  $w \in N(u_1) \cap N(u_2)$  then
10:       $S[w] \leftarrow 0$ 
11:     else
12:       $S[w] \leftarrow 1$ 
13:     end if
14:   end for
15:    $V' \leftarrow V' \setminus (N(u_1) \cup N(u_2) \cup \{u_1, u_2\})$ 
16: end while
17: return  $S$ 

```

set based on empirical tuning to ensure sufficient exploration before reinitialization. The algorithm runs for a fixed number of iterations (*e.g.*, 10 000 in experiments), as described in Section 5 (Experimental results). The feasibility function ensures all solutions are valid QTDRD functions, and the fitness function (sum of labels) guides the search toward minimizing $\gamma_{qtdR}(G)$.

5. HEURISTICS

Three heuristics are proposed to produce the initial population for the genetic algorithm to solve MQT-DRDP. For all the three heuristics, a graph $G(V, E)$ with vertex set $\{v_1, v_2, \dots, v_n\}$ is provided as input. Next, $S[v_1, v_2, \dots, v_n]$ is declared as a solution array and V' is initialized as V . The heuristics are thoroughly described in the following subsections.

5.1. Heuristic 1

In this heuristic, two vertices are randomly selected from the graph and assigned label 2. The common neighbors of these vertices are labeled 0. The neighbors of the first vertex which are not neighbors of the second vertex are labeled 1, and vice versa. The labeled vertices from the vertex set are removed and the vertex set is updated, the process is repeated until the remaining vertex set becomes empty. In this heuristic, every vertex with label 0 is adjacent to two vertices with label 2 and every vertex with label 1 is adjacent to a vertex with label 2. Hence the labelling returned is a valid QTDRDF of G .

The algorithm for this heuristic is given in Algorithm 6, and its functioning is demonstrated with a graph, shown in Figure 2. Initialize V' as $V' = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$. Firstly, vertices v_1 and v_3 are randomly selected, and they are assigned label 2, *i.e.*, $S[v_1, v_3] = 2$. Their common neighbors v_2 and v_4 are labeled 0, *i.e.*, $S[v_2, v_4] = 0$, and the remaining neighbor, *i.e.*, v_6 , is labeled 1, *i.e.*, $S[v_6] = 1$. The labeled vertices are removed from the vertex set, and V' is updated as $V' = \{v_5, v_7, v_8\}$. Vertex v_5 and v_7 are randomly selected and assigned label 2, *i.e.*, $S[v_5, v_7] = 2$ and their common neighbor v_8 is labeled 0, *i.e.*, $S[v_8] = 0$. Now $V' = \emptyset$ and hence S is returned as a feasible solution. The fitness score for this graph G *i.e.*, the sum of the labels of vertices of G is $1 + 0 + 2 + 0 + 2 + 1 + 2 + 0 = 9$.

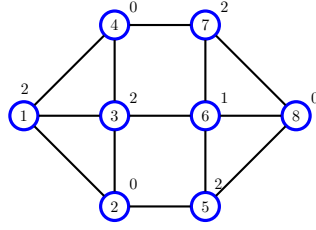


FIGURE 2. Graph labeled by Heuristic 1.

Algorithm 7 Heuristic 2

```

1: Input: A simple and undirected graph  $G = (V, E)$ 
2: Output: Labeling vector  $S$ 
3:  $V' \leftarrow V$ 
4: Sort  $V'$  based on degrees in non-increasing order
5: while  $V'$  is not empty do
6:   Select a vertex  $u$  with the highest degree in  $V'$  by breaking ties arbitrarily
7:   if  $\text{degree}(u) \geq 2$  then
8:      $S[u] \leftarrow 0$ 
9:     for each  $v \in N(u)$  do
10:       $S[v] \leftarrow 2$ 
11:    end for
12:   else if  $\text{degree}(u) = 0$  or  $1$  and  $u$  is adjacent to a vertex labeled 2 then
13:      $S[u] \leftarrow 1$ 
14:   else
15:      $S[u] \leftarrow 2$ 
16:   end if
17:   Remove  $u$  and its neighbors from  $V'$ 
18:   Update degrees of remaining vertices in  $V'$ 
19: end while
20: return  $S$ 
    
```

5.2. Heuristic 2

Vertices are sorted according to their degrees, and one vertex is chosen randomly by breaking ties arbitrarily. If the degree of the selected vertex is greater than or equal to 2, it is labeled 0 and its neighbors are labelled 2. If the degree is 0 or 1, then the label of the neighbors is checked to see if it has a neighbor with label 2. If it is, then the vertex is labeled as 1; otherwise, 2. The labeled vertices from the vertex set are removed, and the degrees of the remaining vertices *i.e.*, unlabeled vertices are updated. The process is repeated until the vertex set is empty.

The algorithm for this heuristic is given in Algorithm 7 and functioning is demonstrated with a graph, shown in Figure 3. Clearly $V' = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$. Vertex v_3 is randomly chosen among the highest degree vertices and labeled 0, *i.e.*, $S[v_3] = 0$; its neighbors v_1, v_2, v_4 , and v_6 are labeled 2, *i.e.*, $S[v_1, v_2, v_4, v_6] = 2$. Now $V' = \{v_5, v_7, v_8\}$. Vertex v_8 is randomly selected among the 3 unlabeled neighbors since it has greatest degree among them and labeled 0, *i.e.*, $S[v_8] = 0$; its unlabeled neighbors v_7, v_5 are labeled 2, *i.e.*, $S[v_7, v_5] = 2$. Now $V' = \emptyset$ and hence the set S is returned as a feasible solution. The fitness score for this graph is 12.

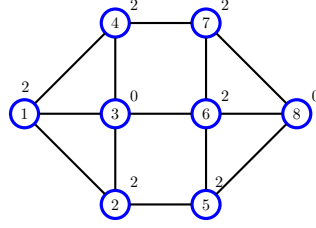


FIGURE 3. Graph labeled by Heuristic 2.

Algorithm 8 Heuristic 3

-
- 1: **Input:** A simple and undirected graph $G = (V, E)$
 - 2: **Output:** Feasible solution S
 - 3: Let $V' \leftarrow V$
 - 4: **while** V' is not empty **do**
 - 5: Sort vertices in V' by degrees in non-increasing order
 - 6: Find an unlabeled vertex u in V' with degree ≥ 1 by breaking ties arbitrarily
 - 7: $S[u] \leftarrow 3$
 - 8: Select one unlabeled neighbor v of u
 - 9: $S[v] \leftarrow 1$
 - 10: **for** each vertex $w \in N(u) \setminus \{v\}$ **do**
 - 11: $S[w] \leftarrow 0$
 - 12: **end for**
 - 13: **for** each vertex x in V' such that $\text{degree}(x)=0$ **do**
 - 14: $S[x] \leftarrow 2$
 - 15: **end for**
 - 16: Remove u and its neighbors from V'
 - 17: Update degrees of remaining vertices in V'
 - 18: **end while**
 - 19: **return** S
-

5.3. Heuristic 3

In this heuristic, similar to Heuristic 2, the vertices are sorted according to their degrees, and a vertex with the highest degree is selected by breaking ties arbitrarily. If the degree of the selected vertex is greater than 0, then it is labeled 3, one of its neighbors is labeled 1, and the remaining neighbors are labeled 0. We label the degree 0 vertices with label 2. These vertices from the vertex set are removed, the degrees of the remaining vertices in the degree set are updated, and the process is repeated for the remaining vertices until the set becomes empty. Finally, we return the labels of the vertices.

The algorithm for this heuristic is given in Algorithm 8, and its functioning is demonstrated with a graph, shown in Fig. 4, $V' = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$. v_3 is chosen among the vertices with the highest degree and labeled 3, *i.e.*, $S[v_3] = 3$. One of its neighbors, v_6 , is then labeled 1, *i.e.*, $S[v_6] = 1$. The remaining neighbors v_1, v_2 , and v_4 are labeled 0, $S[v_1, v_2, v_4] = 0$. Now $V' = \{v_5, v_7, v_8\}$. v_8 is randomly chosen and labeled 3, *i.e.*, $S[v_8] = 3$, one of its neighbors, v_7 , is then labeled 1, *i.e.*, $S[v_7] = 1$. and its neighbor, v_5 , is labeled 0, *i.e.*, $S[v_5] = 0$. Now $V' = \emptyset$ and the algorithm returns S as a feasible solution. The fitness score of the obtained labelling of graph G is 8.

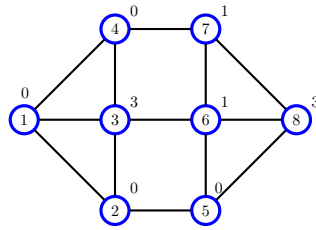


FIGURE 4. Graph labeled by Heuristic 3.

TABLE 1. Standard graphs with existing γ_{qtdR} .

Graph	Optimal value
Path	$n + 1$, if $n > 1$
Cycle	n , if $n \equiv 0 \pmod{2}$ $n + 1$, if $n \equiv 1 \pmod{2}$

TABLE 2. Results for path and cycle graphs.

Graph	No. of vertices	Optimal results	Genetic algorithm	ABC algorithm
Path	5	6	6	6
	16	17	17	17
	20	21	21	21
	26	27	27	27
	46	47	47	47
Cycle	101	102	112	102
	5	6	6	6
	16	16	16	16
	20	20	20	20
	26	26	26	26
Cycle	46	46	46	46
	101	102	102	102

6. EXPERIMENTAL RESULTS

The proposed solutions are implemented in the C++ language, and all experiments are performed on an 11th Gen Intel (R) Core™ i3-1115G4 @ 3.00 GHz with 8 GB of RAM. In [5], the quasi-total double Roman domination number of path and cycle graphs has been obtained and is shown in Table 1. The correctness of the proposed solutions were tested on path and cycle graphs.

We first tested the performance of the proposed genetic algorithm and the proposed ABC algorithm on path and cycle graphs for 5 to 101 vertices. It has been found that the solution produced by our method approaches the optimal one as the number of generations, or iterations, increases. The results obtained at 100 000 generations (for both genetic and ABC) by initiating the population using heuristic 2 are given in Table 2.

The genetic and the ABC algorithm based solutions were evaluated on the Harwell-Boeing dataset, a widely used dataset for graph problems [18, 19]. The experiments involved initializing the complete population using heuristic 1, followed by heuristic 2, and finally heuristic 3 (10000 iterations each) using the proposed genetic algorithm.

TABLE 3. Genetic algorithm results for Harwell–Boeing dataset.

Graphs	Heuristic 1	Heuristic 2	Heuristic 3	Combined	UB
bcsppwr01	43	53	53	50	66
bcsppwr02	73	63	63	63	84
bcsppwr03	167	146	145	145	216
bcsppwr04	370	356	279	356	516
bcsppwr05	658	562	562	588	886
bcsppwr06	2124	1808	1849	1968	2882
bcsppwr07	2379	2006	2038	2174	3198
bcsppwr08	2396	2034	2034	2196	3220
bcsppwr09	2528	2196	2106	2333	3416
ash85	96	94	68	92	150
ash219	336	232	184	230	418
can_24	29	30	20	20	30
can_61	67	106	31	57	72
can_62	84	70	86	82	110
can_96	103	156	56	97	174
can_73	93	96	84	82	128
can_144	153	148	56	150	258
can_161	192	165	100	172	288
can_187	219	191	124	201	354
can_1054	1274	1874	433	1108	2038
curtis54	61	49	46	46	54
dwt_59	63	72	68	68	106
dwt_162	192	248	108	175	306
dwt_310	362	350	218	347	598
dwt_512	653	632	482	639	994
ibm32	32	32	32	30	38
jgl009	4	4	4	4	11
jgl011	4	4	4	4	13
rgg010	4	4	4	4	12
will57	77	50	40	45	72
will199	242	190	139	195	374
lshp_265	320	312	212	318	516
Ishp_406	494	488	334	490	798
Ishp_1009	1238	1229	805	1233	2004
jagmesh_1	1145	1140	771	1143	1858
jagmesh_2	1237	1233	805	1236	2004
pores_1	24	26	24	21	26
1138_bus	1700	1398	1500	1547	2240
494_bus	736	588	670	663	968
662_bus	956	858	794	882	1304
abb313	445	460	194	274	564
arc130	214	248	14	138	162
ash331	484	472	250	313	636
bcsstk01	43	72	34	34	72
bcsstk02	4	82	4	4	82
bcsstk03	135	174	102	128	212
bcsstk05	163	258	66	112	258
bcsstk06	447	728	197	426	784
bcsstk07	445	728	197	446	784

TABLE 3. continued.

Graphs	Heuristic 1	Heuristic 2	Heuristic 3	Combined	UB
bp_1000	1056	1204	496	926	1028
bp_200	1071	1174	568	918	1078
bp_600	1060	1174	530	962	1040
bp_0	1099	1112	632	958	1112
ck400	471	616	324	433	762
dwt_66	75	98	67	75	120
dwt_72	100	84	103	102	134
dwt_87	98	132	64	103	148
dwt_198	241	312	237	229	372
dwt_209	250	332	249	230	384
dwt_221	259	362	256	256	418
dwt_234	339	322	351	305	448
dwt_245	318	328	260	305	464
dwt_307	361	498	189	326	596
dwt_992	1052	1792	316	989	1948
fs_183_1	250	260	148	148	229
fs_183_3	247	260	148	147	229
lnsp_131	186	138	162	161	220
lnsp_511	701	666	540	663	980
nos1	336	272	347	320	464
nos2	1356	1090	1347	1347	1904
nos4	129	130	101	105	186
nos5	557	734	304	526	890
nos6	910	832	757	806	1350
nos7	951	942	647	870	1444
olm100	125	156	88	108	182
olm500	648	786	408	660	982
steam1	307	458	224	225	458
steam3	54	114	28	28	114

6.1. Statistical analysis of heuristics

Descriptive Statistics Summary

Key Observations

- **Heuristic 3 (H3)** yields the best performance, achieving the lowest mean and median fitness values.
- **Heuristic 2 (H2)** exhibits the highest mean fitness, indicating relatively inferior performance.
- Statistical tests confirm H3 significantly outperforms both H1 and H2.

Paired t-Test Results

Correlation Between Heuristics

While all three heuristics are positively correlated in trend, **Heuristic 3 consistently outperforms the others** with statistically significant lower fitness values, making it the most effective strategy under the minimization objective.

According to statistical analysis, the population was initiated by taking 65% of heuristic 3, 20% of heuristic 1, and 15% of heuristic 2 for 10 000 iterations, and the results are shown in the column named ‘Combined’ of Table 3 and ‘ABC’ column of Table 7. The ABC algorithm was also tested on 10 000 iterations. In each iteration, the fitness will be calculated twice and compared to retain the best solution obtained and discard the

TABLE 4. Descriptive statistics for fitness values across heuristics.

Metric	Heuristic 1 (H1)	Heuristic 2 (H2)	Heuristic 3 (H3)
Mean	511.73	524.27	363.85
Median	283.00	312.00	195.50
Std Dev	588.84	555.01	482.20
Min	4.00	4.00	4.00
Max	2528.00	2196.00	2106.00

TABLE 5. Paired t-test comparison of heuristics (lower is better).

Comparison	t-Statistic	p-Value	Better heuristic
H1 vs H2	-0.690	0.492	Similar
H1 vs H3	7.226	3.10e-10	H3 Better
H2 vs H3	5.060	2.77e-6	H3 Better

TABLE 6. Pearson correlation coefficients.

Pair	Correlation
H1 & H2	0.9623
H1 & H3	0.9625
H2 & H3	0.8634

worst. The results in the column Heuristic 1 denote the solution to the MQTDRP for various graphs using the proposed genetic algorithm, initial population was initiated using heuristic 1. Similarly, the columns Heuristic 2 and Heuristic 3 denote the results obtained using the proposed genetic algorithm after generating the original population using Heuristics 2 and 3, respectively. The optimal solution for these graphs is unknown, and no meta-heuristic technique has been provided in the literature to address MQTDRP. We investigated upper limits [5] using the formula $\gamma_{qtdR}(G) \leq 2n - 2\Delta(G) + 2$, where $\Delta(G)$ is the maximum degree of graph G . We examined and compared the performance of the ABC algorithm with the genetic algorithm on the Harwell-Boeing dataset [18, 19]. The results are shown in Table 7. We also tested both solutions on random graphs generated using Barabási-Albert [20, 21], Erdős-Rényi [22, 23], Watts-Strogatz [24, 25], and Kleinberg’s models [26, 27] for 10000 iterations. The results are shown in Tables 8, 9, 10, and 11, where UB stands for Upper Bound.

6.2. Binomial distribution test

In order to determine whether the artificial bee colony optimization (ABC) method outperforms the genetic algorithm (GA), we have performed the sign test.

Null Hypothesis: There is no difference in the performance of GA and ABC for solving the MQTDRP.

Alternative Hypothesis: The ABC algorithm regularly outperforms the GA algorithm in solving the MQTDRP.

We tested on a total of 139 graphs of which 120 resulted in positive signs.

- Number of positive signs (+): 120
- Number of negative signs (-): 4
- Number of ties: 15

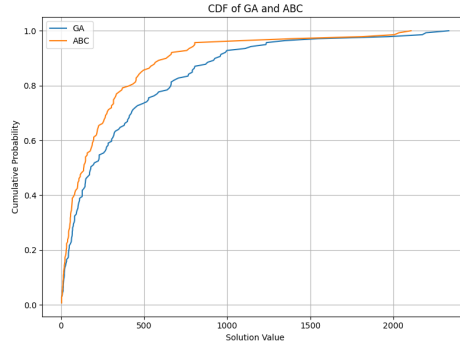


FIGURE 5. CDF curve.

Effective Number of Trials:

$$n_{\text{eff}} = 139 - 15 = 124$$

- Success probability under the null hypothesis (H_0): $p = 0.5$
- Number of successes (k): 120
- Number of trials (n): 124

$$P(X \geq 120) = \sum_{k=120}^{124} \binom{124}{k} p^k (1-p)^{124-k}$$

Given the extremely high number of successes (120 out of 124), indicating a significant difference. When compared to a specified alpha threshold of 0.05, the critical value is $4.560507128 \times 10^{-31}$, which is extremely low. As a result, we reject the null hypothesis, implying a substantial difference in the performance of GA and ABC, and accept the alternative hypothesis, the ABC algorithm consistently outperforms the GA algorithm in solving the MQTDRP.

6.3. CDF analysis of the binomial sign test

To assess the statistical significance of the performance difference between the Artificial Bee Colony (ABC) algorithm and the Genetic Algorithm (GA) on the MQTDRP, we performed a non-parametric sign test. The results showed that the ABC algorithm outperformed GA in the vast majority of instances.

Test Parameters

- Total number of graph instances tested: $n = 139$
- Number of tied outcomes: 15
- Effective number of trials: $n_{\text{eff}} = n - \text{ties} = 124$
- Number of trials where ABC outperformed GA: $k = 120$
- Null hypothesis probability (p) under H_0 : 0.5

CDF Definition

Let the random variable $X \sim \text{Binomial}(n = 124, p = 0.5)$ represent the number of times ABC outperforms GA under the assumption that both algorithms are equally likely to succeed. The **Cumulative Distribution**

TABLE 7. Comparison of genetic algorithm with ABC algorithm for Harwell–Boeing dataset.

Graphs	Genetic algorithm	ABC algorithm	UB	Sign
bcpwr01	50	53	66	-ve
bcpwr02	63	63	84	0
bcpwr03	145	145	216	0
bcpwr04	356	279	516	+ve
bcpwr05	588	560	886	+ve
bcpwr06	1968	1808	2882	+ve
bcpwr07	2174	2006	3198	+ve
bcpwr08	2196	2034	3220	+ve
bcpwr09	2333	2106	3416	+ve
ash85	92	68	150	+ve
ash219	230	184	418	+ve
can_24	20	20	30	0
can_61	57	31	72	+ve
can_62	82	76	110	+ve
can_96	97	56	174	+ve
can_73	82	84	128	-ve
can_144	150	56	258	+ve
can_161	172	100	288	+ve
can_187	201	124	354	+ve
can_1054	1108	433	2038	+ve
curtis54	46	46	54	0
dwt_59	68	68	106	0
dwt_162	175	108	306	+ve
dwt_310	347	218	598	+ve
dwt_512	639	482	994	+ve
dwt_992	998	316	1948	+ve
ibm32	30	32	38	-ve
jgl009	4	4	11	0
jgl011	4	4	13	0
rgg010	4	4	12	0
will57	45	40	72	+ve
will199	195	139	374	+ve
lshp_265	318	212	516	+ve
lshp_406	490	334	798	+ve
lshp_1009	1233	805	2004	+ve
jagmesh_1	1143	771	1858	+ve
jagmesh_2	1236	805	2004	+ve
pores_1	21	16	26	+ve
1138_bus	1547	1398	2240	+ve
494_bus	663	588	968	+ve
662_bus	882	794	1304	+ve
abb313	274	194	564	+ve
arc130	138	14	162	+ve
ash331	313	250	636	+ve
bcsstk01	34	34	72	0
bcsstk02.	4	4	82	0
bcsstk03	128	102	212	+ve
bcsstk05	112	66	258	+ve
bcsstk06	426	197	784	+ve
bcsstk07	446	197	784	+ve

TABLE 7. continued.

Graphs	Genetic algorithm	ABC algorithm	UB	Sign
bp_1000	926	496	1028	+ve
bp__200	918	568	1078	+ve
bp__600	962	530	1040	+ve
bp__0	958	632	1112	+ve
ck400	433	324	762	+ve
dwt___59	68	65	106	+ve
dwt___66	75	67	120	+ve
dwt___72	102	82	134	+ve
dwt___87	103	64	148	+ve
dwt__162	179	108	306	+ve
dwt__198	229	158	372	+ve
dwt__209	230	148	384	+ve
dwt_221	256	136	418	+ve
dwt_234	305	256	448	+ve
dwt_245	305	260	464	+ve
dwt_307	326	189	596	+ve
dwt_992	989	316	1948	+ve
fs_183.1	148	148	229	0
fs_183.3	147	148	229	-ve
lmsp_131	161	138	220	+ve
Insp_511	663	540	980	+ve
nos1	320	268	464	+ve
nos2	1347	1088	1904	+ve
nos4	105	101	186	+ve
nos5	526	304	890	+ve
nos6	806	757	1350	+ve
nos7	870	647	1444	+ve
olm100	108	88	182	+ve
olm500	660	408	982	+ve
steam1	225	224	458	+ve
steam3	28	28	114	0

TABLE 8. Comparison of genetic algorithm with ABC algorithm for Barabási–Albert graphs.

n	m	Genetic algorithm	ABC algorithm	UB	Sign
100	2	145	98	150	+ve
100	5	128	58	128	+ve
100	7	112	48	112	+ve
100	10	94	40	94	+ve
300	2	384	304	516	+ve
300	5	460	176	474	+ve
300	7	410	158	410	+ve
300	10	400	136	400	+ve
500	2	652	462	834	+ve
500	5	764	316	832	+ve
500	7	802	266	802	+ve
500	10	768	228	768	+ve

TABLE 9. Comparison of Genetic algorithm with ABC algorithm for Erdős–Rényi Graphs.

n	p	Genetic algorithm	ABC algorithm	UB	Sign
25	0.2	22	22	36	0
25	0.5	15	12	14	+ve
25	0.8	8	8	8	0
50	0.2	34	30	68	+ve
50	0.5	14	14	38	0
50	0.8	13	10	14	+ve
75	0.2	50	34	110	+ve
75	0.5	24	18	62	+ve
75	0.8	9	8	20	+ve
100	0.2	67	34	138	+ve
100	0.5	19	16	80	+ve
100	0.8	13	8	22	+ve
200	0.2	82	44	288	+ve
200	0.5	44	20	176	+ve
200	0.8	17	10	38	+ve
300	0.2	80	52	442	+ve
300	0.5	48	24	248	+ve
300	0.8	20	12	86	+ve
400	0.2	172	56	596	+ve
400	0.5	45	24	354	+ve
400	0.8	14	12	112	+ve
500	0.2	180	54	746	+ve
500	0.5	74	24	438	+ve
500	0.8	16	14	154	+ve

Function (CDF) of a binomial distribution gives the probability that the random variable takes on a value less than or equal to k :

$$F(k) = P(X \leq k) = \sum_{x=0}^k \binom{n}{x} p^x (1-p)^{n-x}$$

For our sign test, we are interested in computing the one-tailed probability:

$$P(X \geq 120) = 1 - F(119)$$

Computational Result

Using the binomial CDF function, we compute:

$$F(119) = P(X \leq 119) = \sum_{x=0}^{119} \binom{124}{x} (0.5)^{124}$$

$$P(X \geq 120) = 1 - F(119) \approx 0.0$$

TABLE 10. Comparison of genetic algorithm with ABC algorithm for Watts–Strogatz Graphs.

n	m	p	Genetic algorithm	ABC algorithm	UB	Sign
200	5	0.2	284	188	388	+ve
200	5	0.7	284	180	386	+ve
300	5	0.2	422	286	586	+ve
300	5	0.7	412	276	586	+ve
400	5	0.2	578	366	788	+ve
400	5	0.7	560	370	784	+ve
500	5	0.2	706	452	986	+ve
500	5	0.7	682	470	982	+ve
50	5	0.2	70	46	90	+ve
50	5	0.7	70	46	86	+ve

TABLE 11. Comparison of results of Genetic algorithm with ABC algorithm for Kleinberg’s Graphs.

n	k	q	p	Genetic algorithm	ABC algorithm	UB	Sign
50	4	3	0.3	64	58	92	+ve
50	4	3	0.3	62	60	94	+ve
100	4	3	0.3	128	116	190	+ve
100	4	3	0.3	128	116	194	+ve
200	4	3	0.3	266	220	390	+ve
200	4	3	0.3	272	216	388	+ve
300	4	3	0.3	380	350	592	+ve
300	4	3	0.3	402	330	592	+ve
400	4	3	0.3	528	448	790	+ve
400	4	3	0.3	516	452	790	+ve
600	4	3	0.3	790	662	1190	+ve
600	4	3	0.3	790	666	1188	+ve

This result is essentially zero, indicating that 120 or more successes out of 124 trials under the assumption of no performance difference is extremely unlikely.

Interpretation

The p-value from the CDF computation is significantly smaller than the typical significance threshold $\alpha = 0.05$. Therefore, we **reject the null hypothesis** H_0 , concluding that there is statistically significant evidence to support that the ABC algorithm consistently outperforms the GA algorithm in solving the MQTDRP. The CDF analysis reinforces the robustness of our empirical findings by quantifying the improbability of the observed performance results occurring by random chance.

7. CONCLUSION

A genetic algorithm based algorithm and an artificial bee colony optimization based solution are proposed to solve the quasi total double Roman domination problem, which is NP-hard. The performance of both the algorithms is evaluated on the Harwell-Boeing dataset and on the random graphs constructed using the Erdős-Rényi, Barabasi-Albert, Watts-Strogatz, and Kleinberg models. Experiments show that the proposed methods for quasi total double Roman domination problem are efficient. Furthermore, the sign test results show that the ABC based solution performs better than the GA in solving the quasi total double Roman domination problem.

This work serves as a benchmark and other metaheuristics for solving quasi total double Roman domination problem are open.

ACKNOWLEDGMENTS

We would like to thank the reviewers for their insightful comments and constructive suggestions, which significantly improved the clarity and quality of this manuscript. We also appreciate their time and effort in evaluating our work. Their feedback has been instrumental in refining our methodology and presentation.

REFERENCES

- [1] D.B. West *et al.*, Introduction to Graph Theory, vol. 2. Prentice hall Upper Saddle River (2001).
- [2] T.W. Haynes, S. Hedetniemi and P. Slater, Fundamentals of Domination in Graphs. CRC Press (2013).
- [3] E.J. Cockayne, P.A. Dreyer Jr, S.M. Hedetniemi and S.T. Hedetniemi, Roman domination in graphs. *Discrete Math.* **278** (2004) 11–22.
- [4] G. Hao, L. Volkmann and D.A. Mojdeh, Total double roman domination in Graphs. *Commun. Comb. Optim.* **5** (2020) 27–39.
- [5] S. Kosari, S. Babaei, J. Amjadi, M. Chellali and S. Sheikholeslami, Quasi total double roman domination in graphs. *AKCE Int. J. Graphs Comb.* **21** (2024) 171–180.
- [6] A. Khandelwal, K. Srivastava and G. Saran, On roman domination of graphs using a genetic algorithm, in *Computational Methods and Data Engineering: Proc. of ICMDE 2020*, vol. 1. Springer (2020) 133–147.
- [7] J. Greilhuber, S. Schober, E. Iurlano and G.R. Raidl, A simulated annealing based approach for the roman domination problem, in *International Conference on Metaheuristics and Nature Inspired Computing*. Springer (2023) 28–43.
- [8] R.S. Banovoth and K. Kadambari, Roman domination-based spiking neural network for optimized eeg signal classification of four class motor imagery. *Comput. Biol. Med.* **194** (2025) 110397.
- [9] S.N. Chaurasia and A. Singh, A hybrid evolutionary algorithm with guided mutation for minimum weight dominating set. *Appl. Intell.* **43** (2015) 512–529.
- [10] H. Aggarwal and P.V.S. Reddy, Meta-heuristic algorithms for double roman domination problem. *Appl. Soft Comput.* **154** (2024) 111306.
- [11] M.A. Raju and P.V.S. Reddy, Metaheuristic algorithms for solving roman $\{2\}$ -domination problem. *RAIRO Oper. Res.* **58** (2024) 2107–2121.
- [12] J.H. Holland, Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975).
- [13] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA (1989).
- [14] M. Mitchell, An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA (1996).
- [15] D. Karaboga and B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *J. Global Optim.* **39** (2007) 459–471.
- [16] B. Akay and D. Karaboga, A Modified Artificial Bee Colony Algorithm for Real-Parameter Optimization. Elsevier (2012).
- [17] W. Gao and S. Liu, Improved Artificial Bee Colony Algorithm for Global Optimization. Elsevier (2013).
- [18] I.S. Duff, R.G. Grimes and J.G. Lewis, Sparse matrix test problems. *ACM Trans. Math. Softw.* **15** (1989) 1–14.
- [19] I.S. Duff, R.G. Grimes and J.G. Lewis, Users’ guide for the Harwell–Boeing sparse matrix collection (release I). Tech. Rep. RAL-92-086. Rutherford Appleton Laboratory (1992).
- [20] A.-L. Barabási and R. Albert, Emergence of scaling in random networks. *Science* **286** (1999) 509–512.
- [21] R. Albert and A.-L. Barabási, Statistical mechanics of complex networks. *Rev. Mod. Phys.* **74** (2002) 47–97.
- [22] P. Erdős and A. Rényi, On random graphs I. *Publ. Math. Debrecen* **6** (1959) 290–297.
- [23] B. Bollobás, Random Graphs, 2nd edn. Cambridge University Press (2001).
- [24] D.J. Watts and S.H. Strogatz, Collective dynamics of ‘small-world’ Networks. *Nature* **393** (1998) 440–442.
- [25] M.E.J. Newman, The structure and function of complex networks. *SIAM Rev.* **45** (2003) 167–256.
- [26] J. Kleinberg, Navigation in a small world. *Nature* **406** (2000) 845.
- [27] J. Kleinberg, Complex networks and decentralized search algorithms. *Proc. Int. Congress Math.* **3** (2006) 1019–1044.