Optimization Techniques for Multi Cooperative Systems MCTR 1021
Mechatronics Engineering
Faculty of Engineering and Materials Science
German University in Cairo

OPTIMIZATION TECHNIQUES FOR WIND FARM LAYOUTS

By

Team 19

Amr Mohamed Abd El-Monem Ahmed Younis
Mohamed Abdelhameed Mohamed Abdelhameed Elsheikh
Youssef Emad Elshorbagy
Omar Mohamed Farmawy
Mahmoud Osama Mahmoud Elshirbeny

Course Team

Dr. Eng Omar Mahmoud Shehata
Mohamed Salah
Jessica Magdy

December 18, 2023

Abtract-This paper presents a comprehensive study on the optimization of wind farm layouts, a critical aspect of enhancing the efficiency and effectiveness of wind energy production. We delve into various optimization techniques, including Simulated Annealing, Genetic Algorithm, Particle Swarm Optimization, Moth Flame Optimization, and Deep Q-Learning.

Each optimization technique is explored through a series of case studies, where different configurations and parameters are applied to assess their effectiveness in optimizing wind farm layouts. These techniques are evaluated based on criteria such as computational complexity, optimality of solutions, and repeatability of results. The paper's findings offer insightful comparisons and analyses of each technique, providing a nuanced understanding of their respective strengths and limitations in the context of wind farm layout optimization. The study concludes with recommendations for future research and practical applications in the field of wind energy.

Contents

List of Abbreviations

WFLO	Wind Farm Layout Optimization
GA	Genetic Algorithm
PSO	Particle Swarm Optimization
MFO	Moth-Flame Optimization
RL	Reinforcement Learning
DQN	Deep Q-Network
SSWMWD	Single Speed with Multiple Wind Directions
SA	Simulated Annealing
HPSOGA	Hybrid Particle Swarm Optimization and Genetic Algorithm
MSWMWD	Multiple Speed with Multiple Wind Directions
WPP	Wind Power Plants
kW	Kilowatt
MWh	Megawatt Hour
RC-MPGA	Real Coded Multi-population Genetic Algorithm
RLMODE	Reinforcement Learning-based Multi-Objective Differential Evolution
DE	Differential Evolution

List of Figures

Chapter 1

Introduction

Wind energy has emerged as a critical player in the global transition towards sustainable and clean energy sources. Wind farms, designed to harness the kinetic energy of the wind, stand as essential components of this renewable energy revolution. However, the optimization of wind farm layouts poses a multifaceted challenge that demands innovative solutions. In this introductory section, we present the significance of Wind Farm Layout Optimization (WFLO), outline the motivation behind this research, and introduce the structure of this paper.

1.1 Problem Definition

Wind Farm Layout Optimization (WFLO) is the art and science of strategically positioning wind turbines within a designated geographical area to maximize energy production while minimizing operational costs. The primary goal is to minimize the interference of wake effects, ensuring each turbine operates efficiently. In essence, WFLO seeks to answer the question: ”How can wind turbines be optimally placed to harness wind energy effectively?”

1.2 Motivation

The motivation for addressing the WFLO problem is twofold. First and foremost, wind energy stands as a prominent pillar in combating climate change and achieving sustainable energy generation. With escalating concerns over environmental degradation and the finite nature of fossil fuels, optimizing wind farm layouts offers a compelling opportunity to enhance the efficiency of wind energy production.

Secondly, as wind energy continues its upward trajectory, the financial and logistical implications of wind farm deployment have never been more significant. The cost-effectiveness of constructing and maintaining wind farms directly correlates with the layout’s efficiency.

Optimized layouts can reduce construction expenses, increase energy yield, and enhance the industry's overall economic viability.

1.3 Objectives of the Study

This research study aims to address the WFLO problem by evaluating and comparing the efficacy of various optimization techniques. The primary focus of this study is to investigate how Simulated Annealing, Genetic Algorithm, Particle Swarm Optimization, Moth Flame Optimization, and Deep Q-Learning can be leveraged to design efficient and effective wind farm layouts. Each of these optimization algorithms offers a unique approach to solving the problem, and our analysis will shed light on their comparative strengths and weaknesses.

1.4 Research Questions

1. How can the Wind Farm Layout Optimization problem be formulated mathematically?
2. What are the strengths and limitations of Simulated Annealing, Genetic Algorithm, Particle Swarm Optimization, Moth Flame Optimization, and Reinforcement Learning in solving WFLO?
3. Which optimization technique(s) yield(s) the most promising results in terms of energy production, cost-efficiency, and environmental impact?
4. How can the findings of this study contribute to the practical implementation of optimized wind farm layouts?

1.5 Structure of the Paper

This paper is organized as follows:

1. **Literature Review**: An in-depth exploration of existing research and methodologies related to WFLO to establish a comprehensive understanding of the field.
2. **Problem Formulation**: A mathematical formulation of the WFLO problem, outlining the key variables, constraints, and objectives involved.
3. **Optimization Techniques**:
    - Simulated Annealing
    - Genetic Algorithm
    - Particle Swarm Optimization

- Moth Flame Optimization
- Deep Q-Learning

4. **Discussion and Conclusion**: A thorough analysis of the results obtained from each optimization technique, highlighting their respective advantages and limitations. We will conclude by summarizing the key findings and their implications for future research and practical wind farm layout design.

Through this structured approach, we attempt to contribute valuable insights into the optimization of wind farm layouts, facilitating informed decision-making in the wind energy industry and furthering our commitment to sustainable and efficient energy production.

Chapter 2

Literature Review

Wind farm layout optimization (WFLO) is a critical aspect in harnessing maximum potential from wind energy resources, requiring intricate planning to minimize wake losses and maintenance costs while maximizing energy yield. Many algorithms have been employed to tackle this optimization challenge. These algorithms span from the Genetic Algorithm (Genetic Algorithm (GA)), Simulated Annealing (Simulated Annealing (SA)), and Pattern Search (PS) to more advanced methodologies like the Particle Swarm Optimization (Particle Swarm Optimization (PSO)). Each of these algorithms carries distinct characteristics in terms of exploration, exploitation, and convergence to the optimal solution.

In their paper, Qureshi and Warudkar (2023) [4] utilize a method that combines Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) to optimize wind farm configurations. This hybrid algorithm, Hybrid Particle Swarm Optimization and Genetic Algorithm (HPSOGA), leverages both GAand PSO, featuring two phases: PSO for parameter adjustment and GA for solution refinement. While it offers benefits like improved global search and faster convergence, it also introduces increased complexity, the need for parameter tuning, and higher computational costs. Results revolved around two cases: Single Speed with Multiple Wind Directions (Single Speed with Multiple Wind Directions (SSWMWD)) and Multiple Speed with Multiple Wind Directions Multiple Speed with Multiple Wind Directions (MSWMWD). The algorithm was employed and executed 40 times to obtain the best objective function values. In the SSWMWD case, the HPSOGA optimization algorithm placed 39 wind turbines to produce a total power output of 18,126 Kilowatt (kW). The cost per kilowatt-hour (kW) was 1.511 × 10^3, and the optimization process resulted in an objective function value of 0.001511. For the MSWMWD case, employing the HPSOGA optimization algorithm the this context resulted in the placement of 40 wind turbines, yielding a power output of 34,967.52 kW. The cost per kilowatt-hour (kW) was at 0.79891×10^{-3}, and the optimization process achieved a function value of 0.00079891. One potential drawback is the complexity of parameter tuning required for both the Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) phases, which

can be challenging and may affect the algorithm's performance. Moreover, the computational cost of HPSOGA is relatively high, potentially limiting its use for large-scale wind farm layout optimization. The paper also lacks a thorough analysis of the algorithm's sensitivity to initial conditions, impacting its consistency in finding optimal solutions. Addressing these issues and offering insights into their impact would enhance the evaluation of the HPSOGA approach.

Kunakote et al.'s study (2022) [2], specifically focuses on the efficiency of various meta-heuristic algorithms, evaluating twelve algorithms. The study's methodology used solution quality, computational time, and robustness as metrics, though it notably lacks in-depth quantitative data. Among the algorithms, Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Differential Evolution (DE) are highlighted for their superior adaptability and solution quality. However, the research could be perceived as overly computational, with insufficient attention to real-world logistical, installation, and operational challenges of wind farm layout.

Yang and Cho's research (2019) [1] utilize the Simulated Annealing (SA) algorithm in wind farm layout optimization. The paper is notable for its practical approach, applying SA to enhance power efficiency and cost-effectiveness in real-world wind farms. The study successfully demonstrates SA's adaptability and superiority in certain aspects over common algorithms, supported by real data validation. In the first case study, Yang and Cho focus on a scenario with a constant 12 m/s wind speed from the north, aligning turbines accordingly. The algorithm deploys 30 turbines and records a total power of 14,269 kW, 91.756% efficiency, and a fitness value of 0.0015479. In another case which considers 36 wind directions and three wind speeds (8, 12, and 17 m/s) with a different probability of occurrence in each direction, Their layout of 41 turbines resulted in 33,966 kW total power, 88.311% efficiency, and a fitness value of 0.0008263. However, the paper lacks a clear evaluation of the algorithm's performance due to variable problem characteristics. It also needs to address the efficiency-cost trade-off in the proposed model and explore more practical wind scenarios for better practical applicability.

In the paper by Çelik, Yıldız, and Şekkeli (2021) [5], they use Particle Swarm Optimization (PSO) to plan out wind farms. The paper successfully develops a model for wind turbine location selection in wind power plants (Wind Power Plants (WPP)) using the PSO method. It demonstrates the capability of the model to calculate annual power generation with a maximum error of only 0.52%, indicating its accuracy. The PSO method also exhibits good performance in optimizing WPP installation. However, the study is limited by its relatively small solution space, as it considers only 10 wind turbines in 400 settlement locations. To gain more significance, the model should be applied to larger-scale WPP installations. Additionally, while the wake effect is neglected in this study, it should be addressed for a more comprehensive analysis of power output.

The paper by Wu et al. (2019) [6] addresses a critical aspect of onshore wind farm optimization by considering the influence of terrain and wake behavior on wind turbine placement. It

introduces an effective approach that optimizes wind turbine layouts while accounting for topographic variations and wake interactions, leading to a notable improvement in the Levelized Production Cost (LPC) for uneven onshore wind farms. The study's applicability to different topographic conditions demonstrates its potential significance in the field. However, the paper could benefit from further exploration of the proposed method's applicability in larger and more complex wind farm scenarios. Future research may also consider active wake monitoring and the integration of wind farm layout optimization with turbine operational optimization for a more comprehensive approach. Additionally, while the authors acknowledge funding and express gratitude for reviewers and editors, a more explicit discussion of potential conflicts of interest could enhance transparency in research disclosure.

In their study [7], Amine Hassoine et al.(2019), employ a Real Coded Multi-population Genetic Algorithm (RC-MPGA) to optimize the placement of wind turbines in a 4,000,000 m² wind farm. The objective is to maximize power generation while addressing the wake effect, where the placement of one turbine can impact the efficiency of others. The optimization process, analyzed through a MATLAB-based RC-MPGA code, compares various turbine configurations, with the optimal layout achieved after 1274 iterations. It was found that while a configuration of 26 turbines offered higher efficiency per turbine, a setup with 30 turbines increased the overall power output, albeit with a marginal reduction in per-turbine efficiency. The study's significance lies in its comparative analysis with prior genetic algorithm-based research, establishing the RC-MPGA's superior efficiency and cost-effectiveness.

In Clerc's paper (2004) [8], "Discrete Particle Swarm Optimization illustrated by the Traveling Salesman Problem" utilizes an innovative way to do PSO. This approach extends the traditional PSO, typically used for continuous problems, to discrete optimization scenarios. The novelty lies in adapting the PSO principles - where particles representing potential solutions move through the problem space influenced by their own experience and that of their neighbors - to work in discrete spaces. He achieves this by redefining the movement and updating rules of particles in a way that suits discrete problems. Traditional PSO is characterized by particles moving in a continuous space, guided by velocity vectors. In Discrete PSO, Clerc replaces these velocity vectors with probability matrices, which dictate the likelihood of a particle's position changing from one state to another. This adaptation is key to applying PSO to problems where solutions are not represented as points in a continuous space, but as discrete entities like permutations or combinations. The paper not only demonstrates the feasibility of Discrete PSO but also delves into the fine-tuning of its parameters, showcasing its potential in tackling a wide range of discrete optimization problems beyond the specific context of the Traveling Salesman Problem.

In his paper (2015) [3], Mirjalili introduces a new optimization technique based on the navigational behavior of moths. This optimization algorithm, referred to as the Moth-Flame Optimization (MFO) algorithm, is inspired by the transverse orientation mechanism of moths,

where these insects maintain a fixed angle with a distant light source (like the moon) for navigation. The paper presents a twist to this behavior by mathematically modeling the moths' attraction to artificial lights, leading to a spiral flight path around these sources. The MFO algorithm is distinctive in its approach to optimization, where it utilizes a population-based strategy, balancing exploration and exploitation in the search space. It operates by creating and manipulating two primary components: moths and flames. Moths represent the solutions searching the space, while flames are the best solutions found so far. The algorithm employs a logarithmic spiral update mechanism, allowing moths to explore the space around flames, promoting both local and global search capabilities. This novel approach is demonstrated to be highly effective in avoiding local optima, a common challenge in optimization tasks. Being tested on a wide range of benchmark functions, including unimodal, multimodal, and composite functions, the MFO algorithm shows great performance in exploration and exploitation, outperforming several well-known algorithms in many instances. Its efficacy is further validated through application to diverse engineering problems, such as welded beam design, gear train design, and marine propeller design. These applications highlight the MFO algorithm's versatility and capability in handling complex, real-world optimization problems with constrained and unknown search spaces.

Yu and Lu in their paper (2023) [9] present an approach to optimizing the layout of wind farms through a novel algorithm called Reinforcement Learning-based Multi-Objective Differential Evolution (RLMODE). This algorithm integrates Reinforcement Learning (RL) with multi-objective Differential Evolution (DE) to enhance the efficiency and effectiveness of wind farm layout optimization. The use of reinforcement learning allows the algorithm to dynamically and adaptively adjust parameters, thereby balancing the trade-off between exploration and exploitation during the optimization process. The operational principle of RLMODE hinges on the RL technique's capability to dynamically adjust the parameters of the DE algorithm, thereby enhancing its effectiveness in navigating the search space. Specifically, the RL component in RLMODE is employed to optimize the DE algorithm's mutation factor, a critical parameter influencing the balance between global and local search capabilities. By doing so, RLMODE can efficiently explore the search space while maintaining a balance between diversification (exploration) and intensification (exploitation). The algorithm utilizes a tournament-based mutation operator, which selectively focuses the search process around promising solutions, accelerating convergence. Additionally, the RLMODE algorithm incorporates non-dominated and crowding distance sorting techniques to refine the selection of optimal solutions. This aspect ensures that the algorithm not only finds solutions that are viable in terms of the objectives but also maintains a diverse set of solutions, a critical aspect in multi-objective optimization problems.

Dong, Xie, and Zhao presented in their paper (2022) [10] a comprehensive review of the evolving landscape of wind farm control technologies. The paper examines the complexities of wind farm control, driven by the aerodynamic interactions among wind turbines and the variability of environmental conditions. Recognizing the need for advanced control technologies

in this sector, especially for large-scale offshore wind farms, the paper provides an analysis of various control strategies ranging from classical methods to reinforcement learning RL techniques. The authors explore different control objectives, including layout optimization, power generation maximization, fatigue load minimization, and power reference tracking. They emphasize how these objectives aim to enhance operational efficiency, profitability, and reduce maintenance costs in wind farms. A significant focus of the paper is on the transition from traditional model-based and model-free control approaches to RL-based methods, which have emerged as a promising solution to the challenges faced by conventional techniques. The authors detail how RL, is suited for the complex, dynamic environments of wind farms. RL-based methods, as highlighted in the paper, are capable of handling high system complexities and offer robust adaptability, making them a valuable asset in the pursuit of optimized wind farm operations. The paper also discusses the limitations of current strategies and suggests future research directions. It underscores the need for improved robustness against rapid environmental changes and the reduction of computational costs for scalability. The authors suggest that handling the stochastic nature of environmental conditions and ensuring safety under extreme conditions remain key challenges for future research. This document serves as a vital resource for researchers and practitioners in the field, offering insights into the current state and future prospects of wind farm control technologies.

Chapter 3

Problem Formulation

3.1 Problem Overview

Wind farm layout optimization (WFLO) is still a complex problem in the field of wind energy. Wind farm layout optimization is the process of strategically arranging wind turbines within a designated area to maximize energy production and efficiency. The goal is to find the most efficient configuration that minimizes the impact of factors like turbine interference, wind speed variations and directions, and environmental considerations while meeting cost constraints. The biggest affector of farm efficiency is the wake effect. The wake effect refers to the decreased wind speed and turbulence downstream of a wind turbine caused by the extraction of kinetic energy by the turbines in its path. This phenomenon can significantly reduce the energy output of downstream turbines, making it essential to optimize wind farm layouts to minimize this effect.

The primary objective is to design the wind farm layout that maximizes energy production (power output) while minimizing the overall cost. This will become clear as we go through the design of the objective function. As was mentioned, to maximize power output, it's crucial to minimize the wake effect. Downstream turbines receive less wind energy due to the wake of the turbines in front. The number of the turbines used must be considered to minimize the cost incurred. The cost per unit of power (e.g., $/Megawatt Hour (MWh)) is a critical factor. This includes costs for turbine procurement, installation, and maintenance.

There are also multiple constraints in the environment of the problem that must be taken into consideration. One such constraint is needing to avoid sensitive areas or unusable areas due to topography. This creates certain dead zones where the placement of wind turbines isn't possible further complicating the layouting process. In addition to this wind turbine spacing needs to be handled to prevent interference as well as safety and operational concerns.

Deciding the number of wind turbines and where to place them is a complex task with mul-

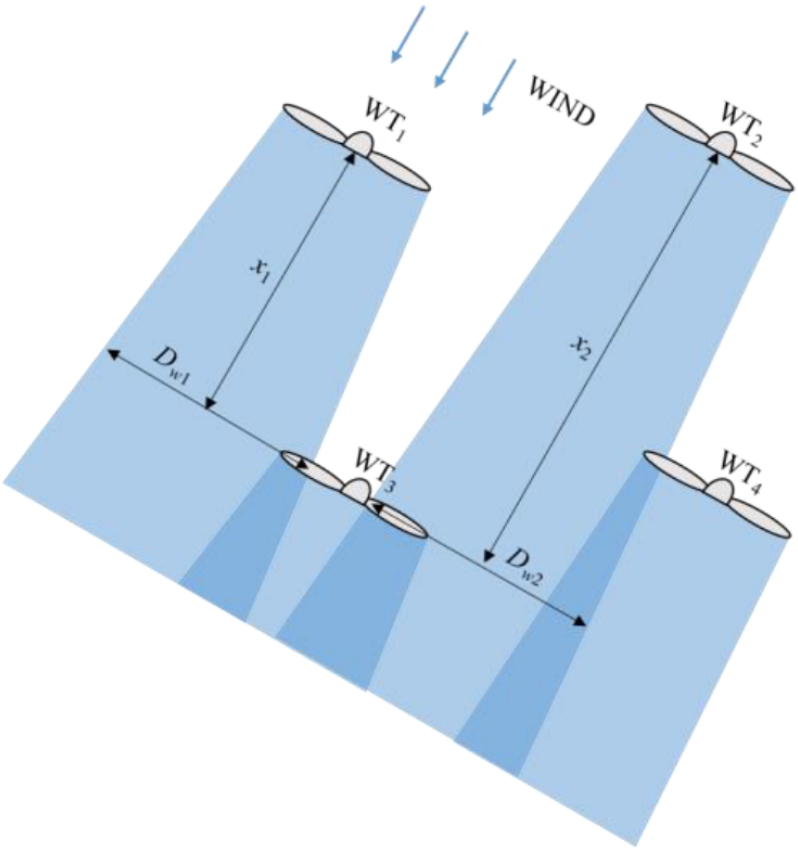

Figure 3.1: Illustration of the wake effect [1]

tiple variations and possible outcomes. For example, even for a low number of WTs (< 30), there might exists over 10^{44} unique solutions, Gualtieri (2019) [11]. Thus conventional numerical methods may prove futile in handling this problem. Therefore, we aim to apply stronger approaches, namely meta-heuristic algorithms to discover the best possible solution for this problem.

3.2 Problem Description

For our problem, only one type of turbine will be considered. We will consider the optimization of the layout in an $n * m$ cells grid where n and m are user defined sizing factors. Each cell in the grid is as large as the turbine diameter. Then the layout optimization will be considered over 36 different wind directions with constant speed to generate a good layout that can operate in multiple wind conditions. Each wind direction will be given a weight to simulate its frequency. A higher frequency means that this particular wind direction occurs more often and thus is

more important for the optimization. During layouting, dead zones will be considered when generating the solutions. For our modelling of the wake effect, we will be using a variation of the Jensen single wake model (1983) [12] approach presented by Qureshi and Warudkar (2023) [4] due to its simplicity and relative accuracy in capturing the effect of wake on power generated.

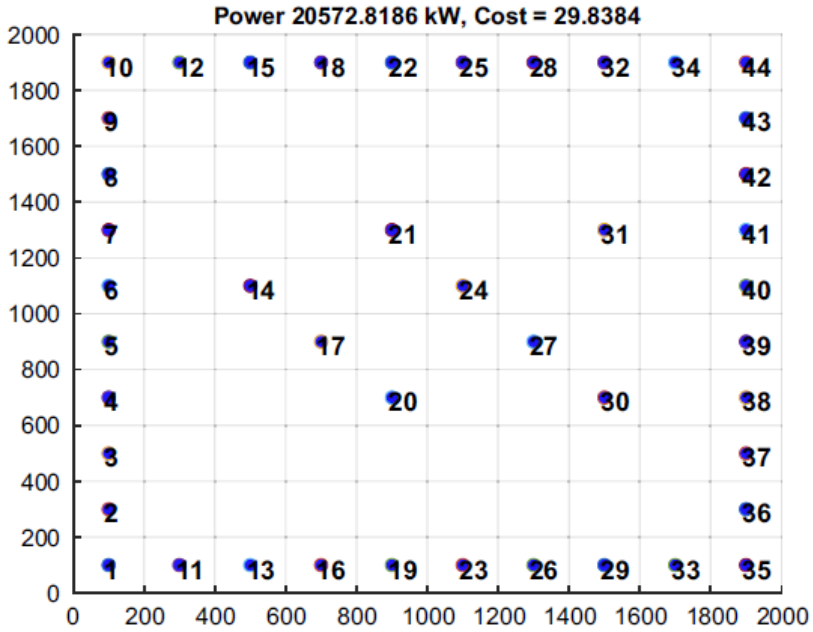

Figure 3.2: Example of a result for the WFLO problem [2]

3.2.1 Objective Function

For the derivation of the objective function, we have 2 aspects that need to be optimized: Total power generated, cost of procurement and maintenance. To explain the power calculation the single wake model has to be explained. It is particularly effective for offshore wind farms and flat terrain far-wake zones due to its simplicity and quick calculations.

The model assumes that the remaining wake behind a wind turbine can be considered a turbulent wake if the near wake is ignored. The size of the wake spread behind the wind turbine rotor increases linearly with the downstream distance (X). Inside the wake, the wind speed is assumed to be identical in the radial direction. The model is derived based on the law of conservation of mass, taking into account parameters such as wind speed behind the rotor (V_1), turbine rotor radius (r_r), downstream wake radius (r_w), freestream wind speed (V_0), and wake wind speed at a downstream distance X (V). The relation between the velocities is as follows :

$$\pi r_r^2 V_1 + \pi \left(r_w^2 - r_r^2\right) V_0 = \pi r_w^2 V \tag{3.1}$$

The velocity (V_1) at a downstream distance X can be calculated using the following equation :

$$\frac{V_1}{V_0} = 1 - \left(1 - \frac{1}{3} * \frac{V}{V_0}\right) \left(\frac{r_r}{r_r + \alpha X}\right)^2 \tag{3.2}$$

where α is the wake decay constant and we chose 0.075 as this is the recommended value for onshore wind farms.

In summary, the single wake model is a practical tool for estimating the wake characteristics behind wind turbines and is commonly used in wind farm layout optimization and will be the basis for our wake calculations. The reason we need this model is it will be used to check for any overlap between the turbines' wake and other turbines. This will affect the total power calculated for the farm. We use the model to calculate the velocities for each wind turbine while accounting for the wake from the other turbines.

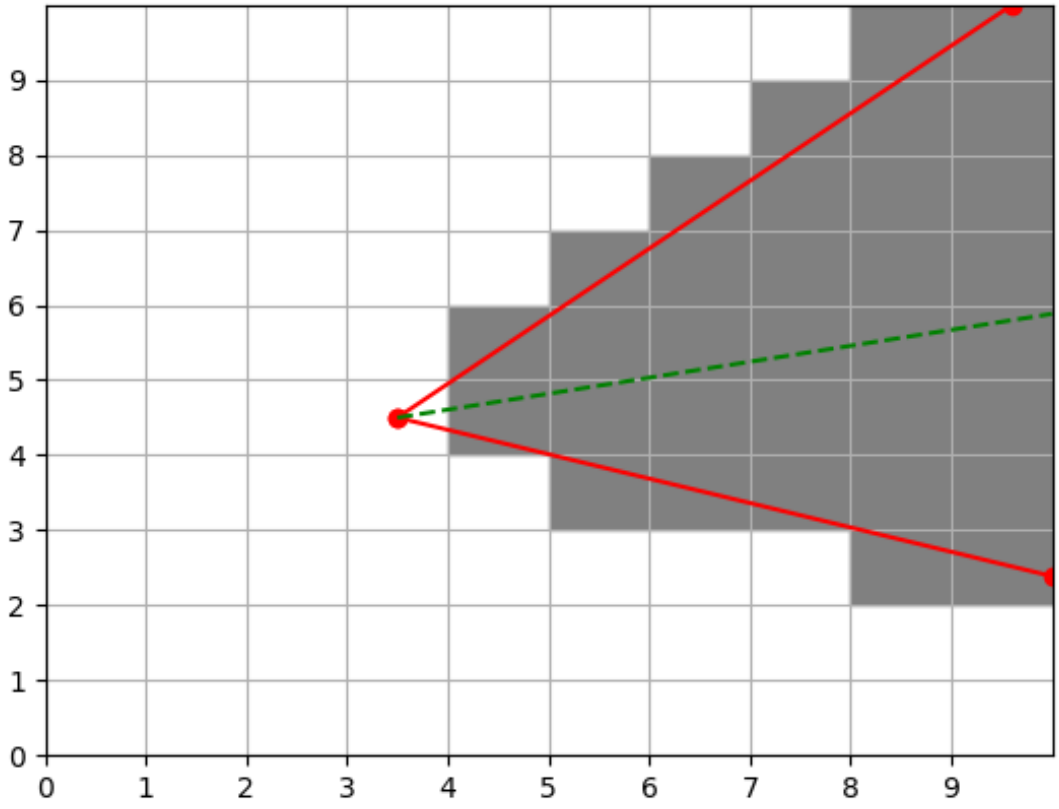

Figure 3.3: Example of wake area. The red lines represent the affecting cone and the green line is the wind direction

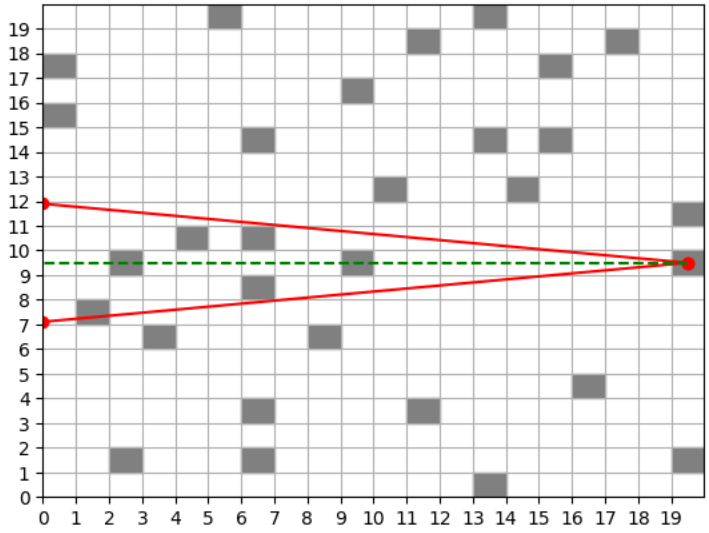

Figure 3.4: Simulation on a 20x20 grid with a wind direction of west and 30 turbines

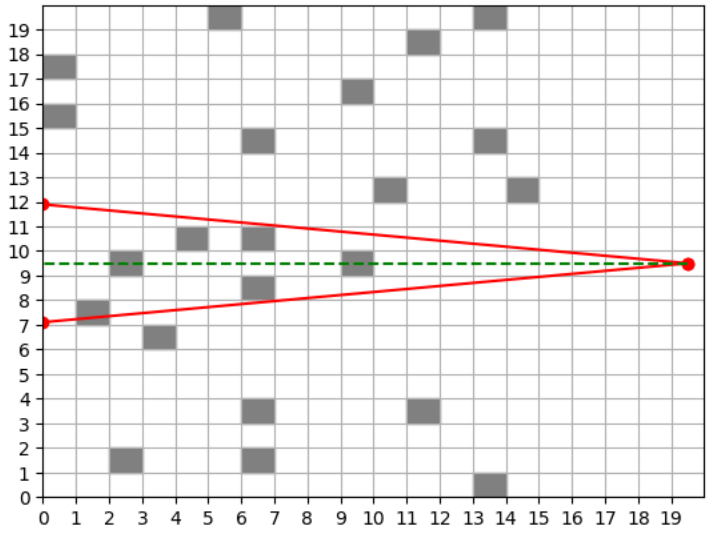

Figure 3.5: Turbines affected by wake

Figure (2.3) shows an example of a wind turbine wake area affecting the cells marked in grey. Figure (2.4) shows an example configuration of wind turbines with a wind direction of west while figure (2.5) shows marked in grey the cells that are affected by some wake from another turbine according to this wind direction. The power of the wind farm at a specific direction W is given by:

$$\mathrm{P}_W = \sum_{i}^{N_{total}} 0.3V_i^3 \tag{3.3}$$

where V_i is the velocity at the ith-turbine and N_{total} is the total number of turbines. Then the total average power can be calculated as the weighted sum of all the powers with the weights being the frequency of each wind direction:

$$\text{Power}_{\text{total}} = \sum_{i=1}^{N_D} \text{P}_{Wi} \cdot W_{Di} \tag{3.4}$$

where N_D is the number of wind direction configurations under consideration, P_{Wi} is the power at the ith wind configuration, and W_{Di} is the weight of the ith direction.

For the calculation of the cost of procurement and maintenance, a model was presented by Qureshi and Warudkar (2023) [4] that captures this relation according to the number of wind turbines being placed :

$$\text{Cost}_{\text{P}} = N_{\text{total}} \left(\frac{2}{3} + \frac{1}{3} e^{-0.00174 N_{\text{total}}^2} \right) \tag{3.5}$$

This model suggests that up to a certain point, buying more turbines is more cost effective due to bulk discounts and hence the exponential term.

Finally the total objective function to be minimized is the aggregation of all the previously mentioned components :

$$\text{Objective Function } = \frac{\text{Cost}_{\text{P}}}{\text{Power}_{\text{total}}} \tag{3.6}$$

3.2.2 Decision Variables

There are two main decision variables in our problem that directly affect the fitness computation :

- N_{total} : An integer value representing the total number of wind turbines to be erected in the farm. This value contributes to both the cost and power of the farm and selecting the correct number of turbines is key to reaching the optimal cost/power ratio for the wind farm.
- Turbine Layout : A 1-dimensional array M representing the turbines placed. Each element is a pair representing the cell currently being occupied by a turbine. The total number of pairs corresponds to the value of N_{total}. This decision variable will directly influence the calculation of the power through the wake model as well as the cost of the area. Better layouts that minimize wake and area will lead to an overall better fitness value.

3.2.3 Constraints

There are five main constraints that we will consider in our problem. We chose to model these constraints in specific as they most closely represent real life problems and the constraints that may be encountered during layouting:

- Grid Dimensions : Two integers (n, m) representing the dimensions of the grid under consideration for layout optimization. The total number of available cells would be :

$$\text{Cells}_{total} = n * m \tag{3.7}$$

  Any turbine's coordinates x_t and y_t must lie within this $n * m$ grid.

- Spacing Distance : An integer denoting the minimum spacing required between each wind turbine in all 8 directions. This was chosen as wind farms are required to have spacing between each turbine for operational and safety concerns. The spacing was chosen to be 3 cells as this is sufficient space between the turbines (each cell is as big as the diameter of a turbine). For an array M representing the layout with pairs $M_{i,j}$:

$$[M_{i,j}] \Rightarrow [\neg M_{i\pm1,j} \wedge \neg M_{i,j\pm1} \wedge \neg M_{i\pm1,j\pm1}] \tag{3.8}$$

  where the negation of $M_{i,j}$ means the pair cannot exist in the array.

- Dead Zones : An array of pairs, each representing the coordinates of a cell. These cells are marked as unavailable for deployment. These were added to model the possible obstacles that could be encountered during layouting including topographical formations.

$$\text{For every element } Z \text{ in } D : \neg M_{Z_y,Z_x} \tag{3.9}$$

  where D is the array of dead zones.

- Maximum number of turbines : An integer that specifies the maximum available turbines that can be used. This constraint was added in case the number of turbines available to be deployed was limited. This could be a consideration in the case there is a given budget for the wind farm. This value is bounded by the maximum number of turbines that can be fitted into the grid while also taking into account the spacing distance and the dead zones.

$$0 < N_{max} < \lfloor \frac{n}{SpacingDistance + 1} \rfloor * \lfloor \frac{m}{SpacingDistance + 1} \rfloor - length(D) \tag{3.10}$$

  where n and m are the grid dimensions and D is the dead zone array.

- Minimum Operational Power Coefficient $MOPC$: A factor that prevents any single wind direction from dominating over others. It is defined as a fraction that ensures the power

generated at any given wind direction is not less than the average generated power across all directions multiplied by this coefficient. This guarantees that no one wind direction is favored over the others and that a minimum operational power level is maintained under all conditions. This constraint is described mathematically as:

$$\min_{\text{wind directions}} (\text{power}) > \text{power}_{\text{avg}} \times MOPC \tag{3.11}$$

Chapter 4

Simulated Annealing

4.1 Algorithm

Simulated annealing is a probabilistic technique used for finding an approximate solution to an optimization problem. It is particularly useful for problems where the search space is large, complex, or poorly understood. Simulated annealing is inspired by the process of annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects.

In the context of optimizing wind farm layout, simulated annealing helps in finding an optimal or near-optimal configuration that maximizes power output or minimizes cost, or in this case, maximizes the ratio of power output to cost. Here's how the process was implemented in the project:

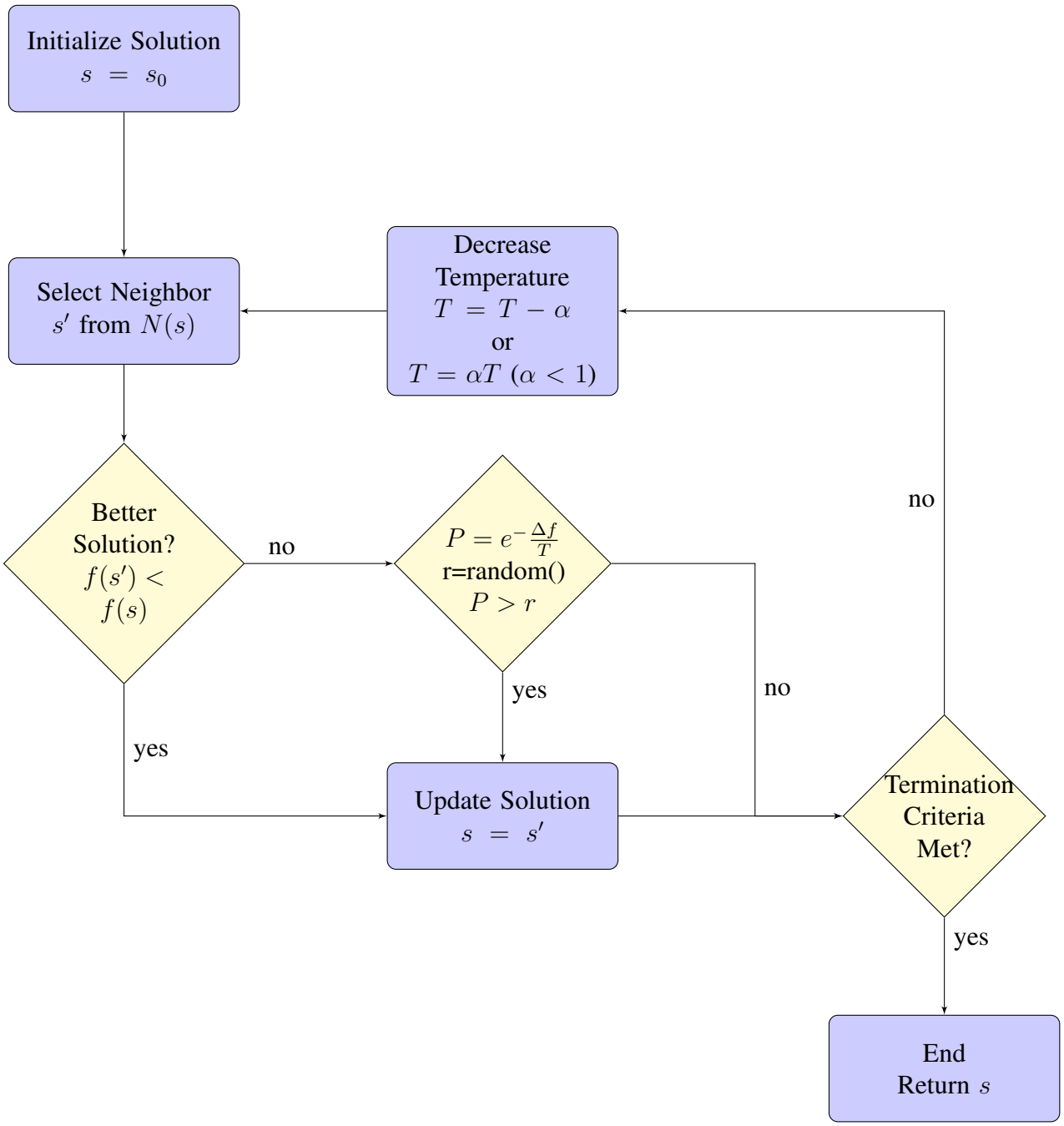

Figure 4.1: Flow chart of the simulated annealing algorithm

4.1.1 Initialization

1. **Setting Parameters:** The algorithm begins with an initial temperature and cools down to a final temperature using a specified decrement. This mimics the metallurgical process of cooling, with the system becoming less dynamic as the temperature decreases.

2. **System Configuration:** The wind farm layout is represented as a grid of $N \times N$ cells, with certain "dead cells" where wind turbines (WT) cannot be placed.
3. **Initial Solution:** A random initial solution is generated by placing wind turbines in random locations within the grid, excluding the dead cells.

4.1.2 Iterative Process

1. **Neighbor Solution Generation:** The algorithm randomly selects one of three methods to generate a new solution:
   - Removing a wind turbine.
   - Adding a wind turbine in a random location.
   - Moving an existing wind turbine to a new location.

   These changes are made while respecting the constraints of the grid space and dead cells.
2. **Solution Evaluation and Acceptance Criterion:** The new configuration's objective function value is compared with the current one. If the new solution is better (i.e., it has a lower objective function value), it is automatically accepted. If not, the algorithm calculates the probability of accepting the worse solution, which depends on the current temperature and the difference in the objective function values. This probability is calculated using the Boltzmann-Gibbs distribution in thermodynamics, encouraging exploration of new solutions at higher temperatures and exploitation of good solutions at lower temperatures.
3. **Cooling and Convergence:** The temperature of the system is reduced in each iteration by the step variable, and the process repeats. As the temperature decreases, the algorithm becomes less likely to accept worse solutions, thereby converging to an optimal or near-optimal solution. The process continues until the system cools to the final temperature.
4. **Tracking the Best Solution:** Throughout its run, the algorithm keeps track of the best solution encountered so far, ensuring that this information is preserved even if subsequent iterations produce inferior results.
5. **Termination:** The algorithm terminates when the temperature decreases to the final temperature through scheduling or when an artificial max number of iterations is imposed and reached.

4.1.3 Conclusion

Through this approach, simulated annealing navigates the search space. By occasionally accepting worse solutions, it avoids being trapped in local minima, an issue common in gradient-

based optimization methods. This characteristic is especially beneficial in complex problems such as ours, where the search space is vast and the optimal configuration is not intuitively evident.

4.2 Case Study 1: 15x15 grid with linear cooling schedule

In this test case, we're working with a 15x15 grid. Additionally, we have marked specific cells as "dead" within the grid, where no wind turbines can be placed. These cells are marked black in the figures. (See Figure 7.3) We initialize the algorithm with an initial temperature of 500 and iteratively decrease it using a linear temperature decay function with a temperature reduction factor of 1. The number of iterations per temperature step is set to 2, and the algorithm will run for a maximum of 500 iterations. This test case examines the impact of the linear temperature decay function on the algorithm's convergence and solution quality.

- **Grid Size:** 15x15
- **Dead Cells:** [(2,2), (12,2), (2,12), (12,12)]
- **Initial Temperature:** 500
- **Final Temperature:** 0
- **Iterations per Temperature Step:** 2
- **Maximum Number of Iterations:** 500
- **Temperature Reduction Factor (α):** 1
- **Scheduling Function:** Linear Temperature Decay Function

$$T = T - \alpha$$

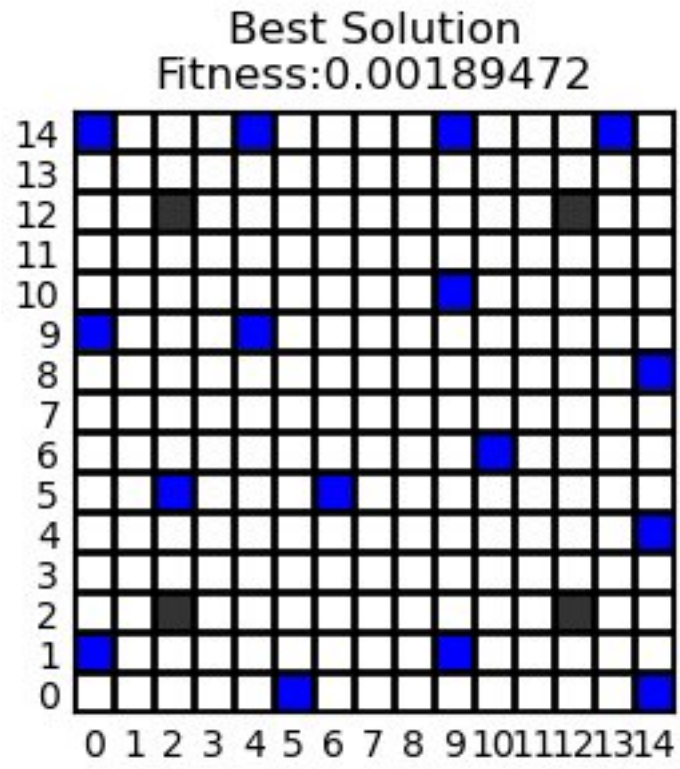

Figure 4.2: Best solution found by SA in case study 1

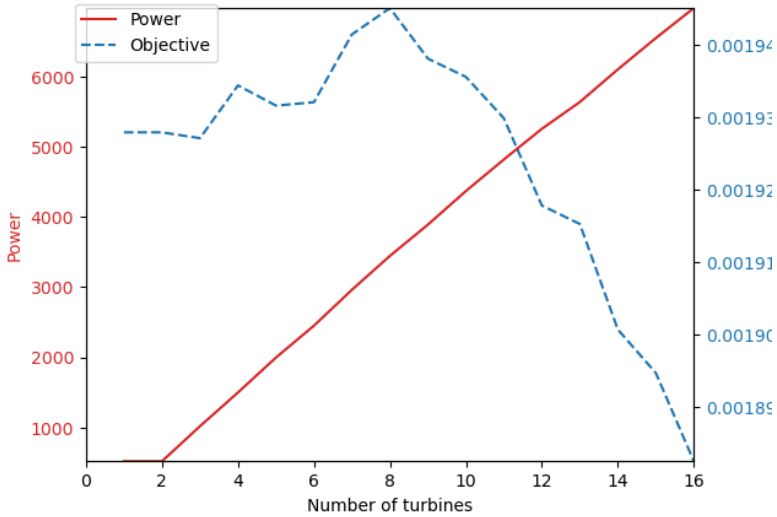

Figure 4.3: Comparison of Power Generation and Fitness Value with Number of Turbines in Case Study 1

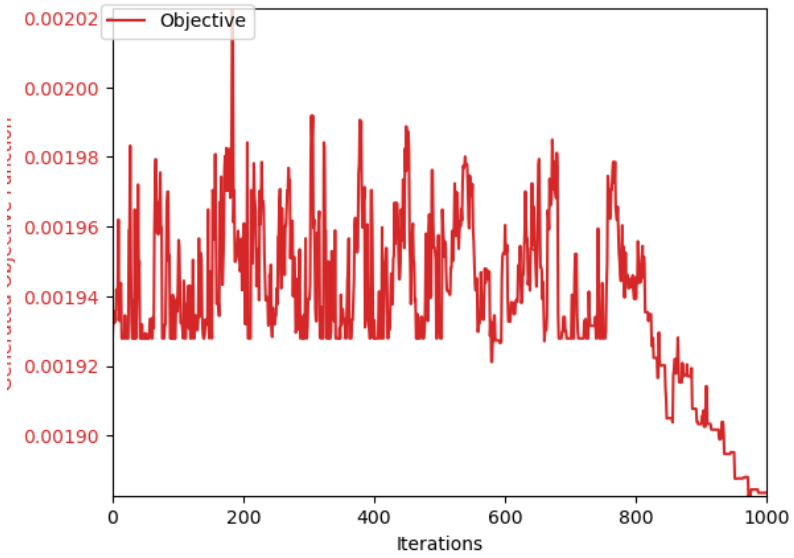

Figure 4.4: Current Fitness Over Iterations in Case Study 1

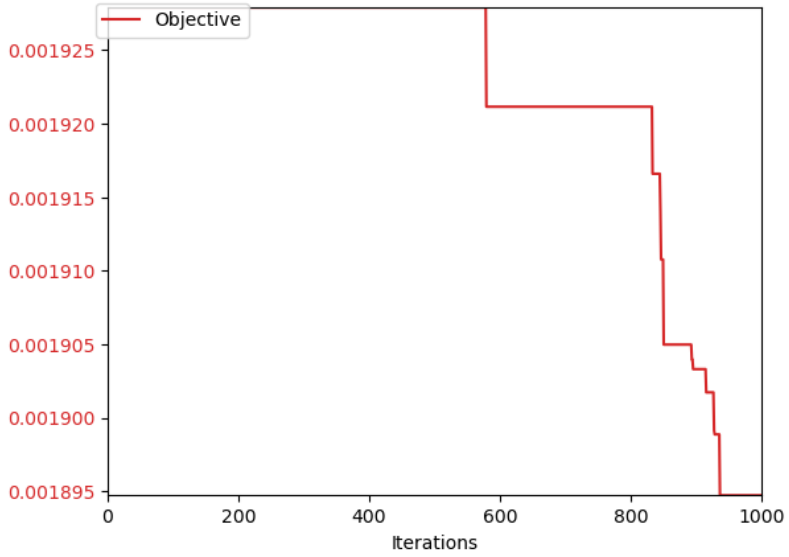

Figure 4.5: Best Fitness Over Iterations in Case Study 1

When comparing the value of the best solution to known benchmarks, our implementation scored 0.00189 while Kunakote et al.'s study (2022) [2] scored 0.00140. These are good results especially when considering the added constraints like the spacing distance and the dead zones that further complicate the layouting as opposed to the grid studied in the paper. This is also the reason why the final layout is more random and much less uniform than the one found in

the paper. Furthermore the power generated from the number of turbines deployed aligns well with the power generated in Qureshi and Warudkar's study (2023) [4]. We deploy 16 WTs in this case generating around 7.5kW giving a ratio of 0.468kW/WT. This aligns well with the paper as they had a ratio of 0.461kW/WT.

4.3 Case Study 2: 20x20 grid with linear cooling schedule

In this test case, we're working with a 20x20 grid. We have marked more cells as "dead" within the grid, as the it is larger in size. We initialize the algorithm with an initial temperature of 500 and iteratively decrease it using the same linear temperature decay function and temperature reduction factor as test case 1. The number of iterations per temperature step is set to 2, and the algorithm will run for a maximum of 500 iterations.

- **Grid Size:** 20x20
- **Dead Cells:** [(3,2),(4,2),(3,3),(4,3),(15,2),(16,2),(15,3),(16,3),(3,16),(4,16),(3,17), (4,17),(15,16),(16,16),(15,17),(16,17)]
- **Initial Temperature:** 500
- **Final Temperature:** 0
- **Iterations per Temperature Step:** 2
- **Maximum Number of Iterations:** 500
- **Temperature Reduction Factor (α):** 1
- **Scheduling Function:** Linear Temperature Decay Function

$$T = T - \alpha$$

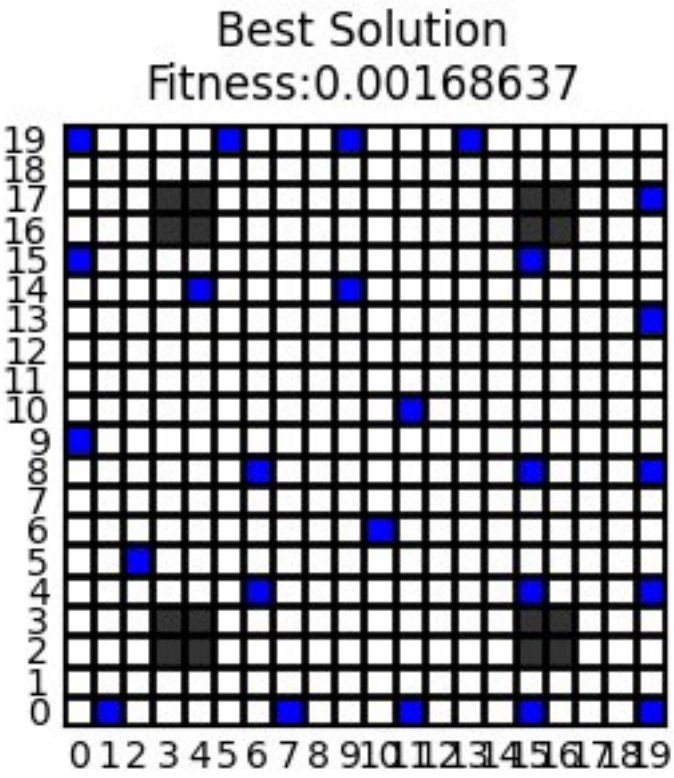

Figure 4.6: Best solution found by SA in case study 2

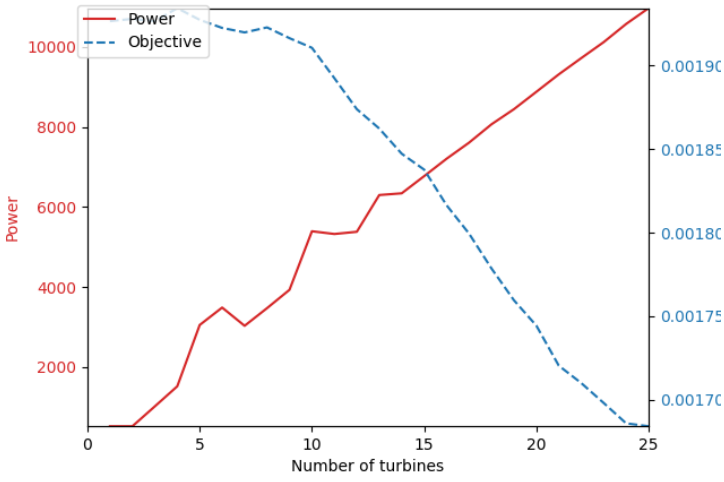

Figure 4.7: Comparison of Power Generation and Fitness Value with Number of Turbines in Case Study 2

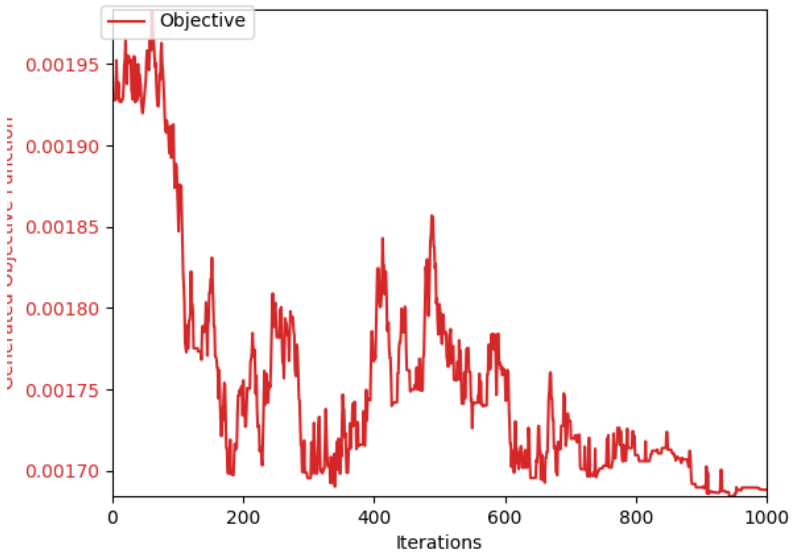

Figure 4.8: Current Fitness Over Iterations in Case Study 2

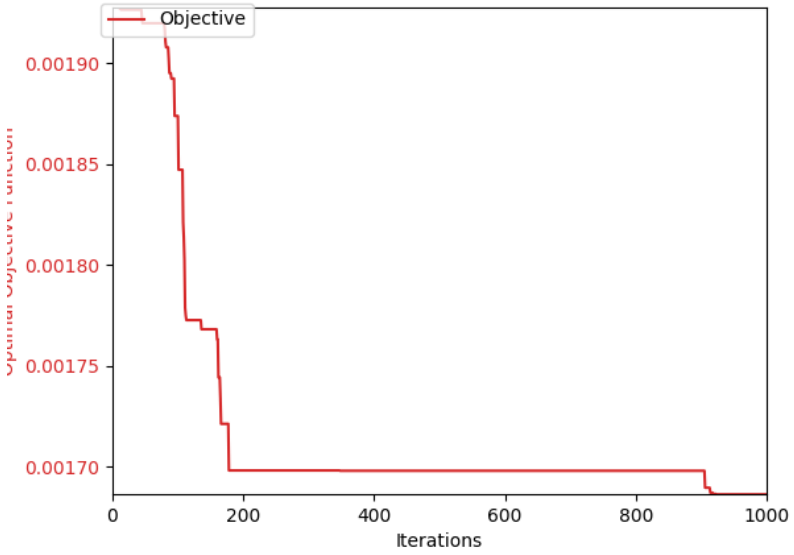

Figure 4.9: Best Fitness Over Iterations in Case Study 2

For this case, the difference became smaller in the fitness. We scored 0.00168 as opposed to 0.00140 reported by Kunakote et al.'s study (2022) [2]. We had 25 WTs producing 11kW giving 0.44kW/WT which is an improvement.

4.4 Case Study 3: 25x25 grid with linear cooling schedule

In this test case, we're working with a 25x25 grid. We have marked more cells as "dead" within the grid. We initialize the algorithm with an initial temperature of 500 and iteratively decrease it using the linear temperature decay function and a temperature reduction factor of 1. The number of iterations per temperature step is set to 2, and the algorithm will run for a maximum of 500 iterations.

- **Grid Size:** 25x25
- **Dead Cells:** [(5,5),(5,6),(6,5),(6,6),(5,18),(5,19),(6,18),(6,19),(18,5),(19,5),(18,6) ,(19,6),(18,18),(18,19), (19,18),(19,19),(7,7),(7,6),(7,5),(7,18),(7,19),(18,7),(19,7) ,(5,7), (6,7),(5,17),(6,17),(7,17),(17,5),(17,6),(17,7),(17,17),(17,18),(17,19),(18,17),(19,17)]
- **Initial Temperature:** 500
- **Final Temperature:** 0
- **Iterations per Temperature Step:** 2
- **Maximum Number of Iterations:** 500
- **Temperature Reduction Factor (α):** 1
- **Scheduling Function:** Linear Temperature Decay Function

$$T = T - \alpha$$

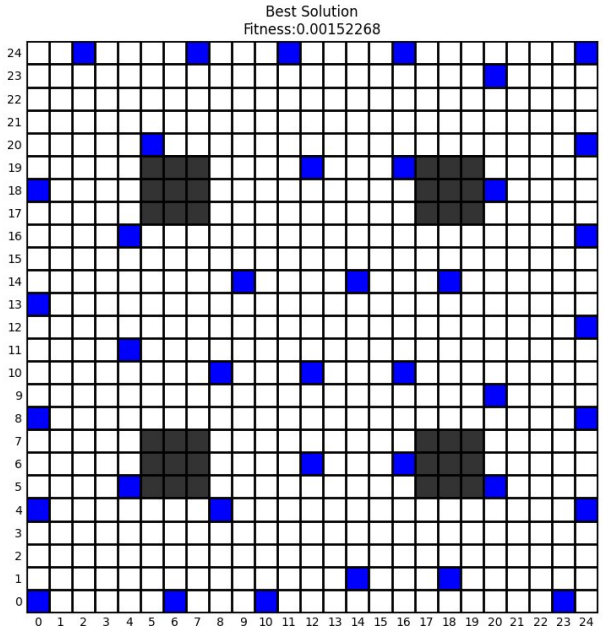

Figure 4.10: Best solution found by SA in case study 3

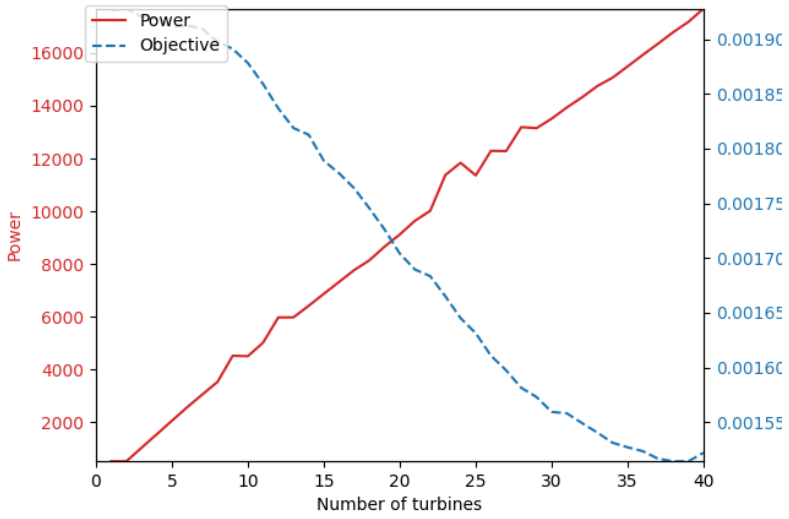

Figure 4.11: Comparison of Power Generation and Fitness Value with Number of Turbines in Case Study 3

For this case, the difference became even smaller in the fitness. We score 0.00152 as opposed to 0.00140 reported by Kunakote et al.'s study (2022) [2]. We had 39 WTs producing 18kW giving 0.461kW/WT which is identical to the benchmark. This is a trend that was noticed throughout our testing: this algorithm's performance scales with the size of the problem.

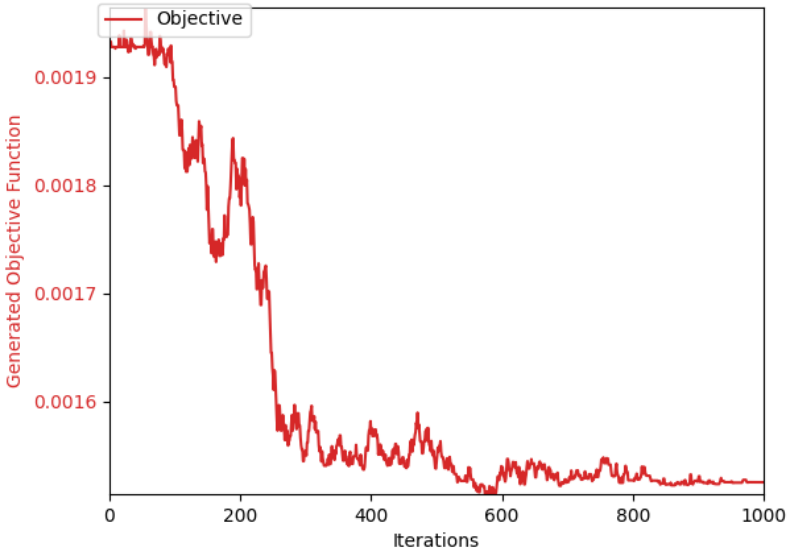

Figure 4.12: Current Fitness Over Iterations in Case Study 3

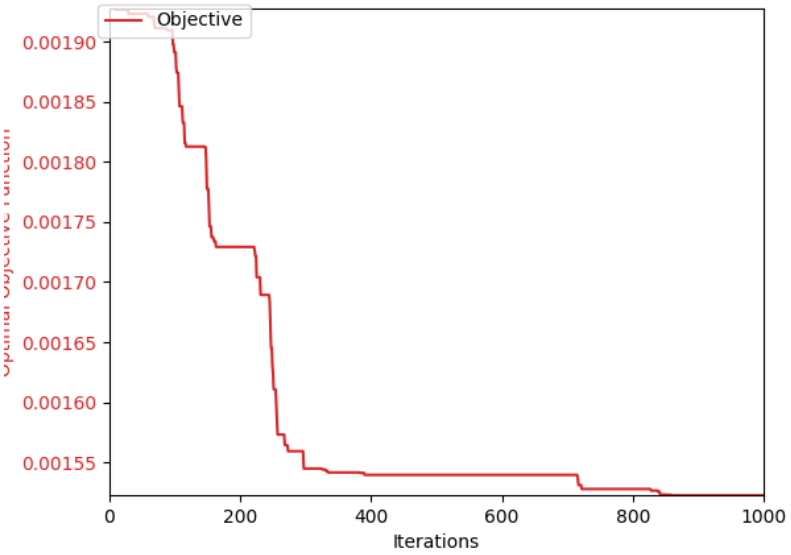

Figure 4.13: Best Fitness Over Iterations in Case Study 3

4.5 Case Study 4: 15x15 grid with geometric cooling schedule

For this test case, we introduce a change in the temperature decay function. This time, we employ a geometric temperature decay function instead of a linear one. The initial temperature

is 1000, with the same 2 iterations per temperature step, a maximum of 500 iterations and a temperature reduction factor of 0.95. We'll once again work with a 15x15 grid, and the specified "dead cells" are included. This experiment aims to investigate how a geometric decay function impacts the performance and results of the simulated annealing algorithm in the same problem context.

- **Grid Size:** 15x15
- **Dead Cells:** [(2,2), (12,2), (2,12), (12,12)]
- **Initial Temperature:** 1000
- **Final Temperature:** 0
- **Iterations per Temperature Step:** 2
- **Maximum Number of Iterations:** 500
- **Temperature Reduction Factor (α):** 0.95
- **Scheduling Function:** Geometric Temperature Decay Function

$$T = T * \alpha$$

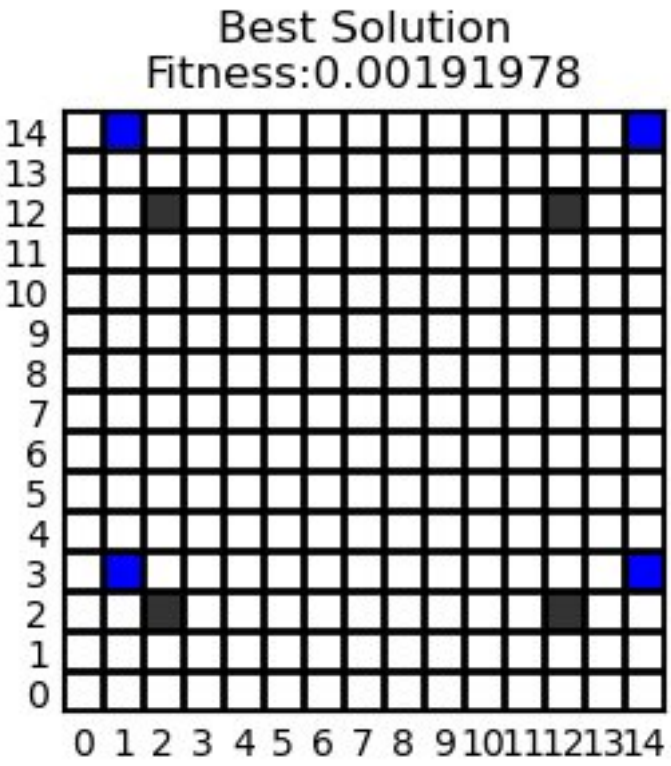

Figure 4.14: Best solution found by SA in case study 4

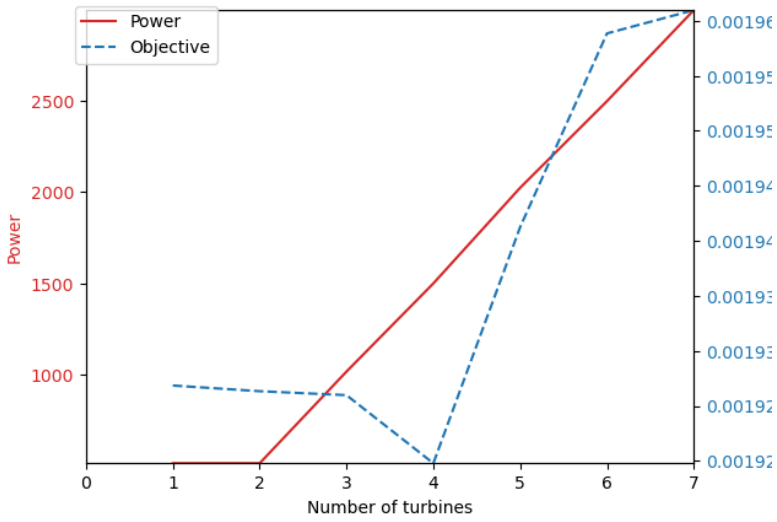

Figure 4.15: Comparison of Power Generation and Fitness Value with Number of Turbines in Case Study 4

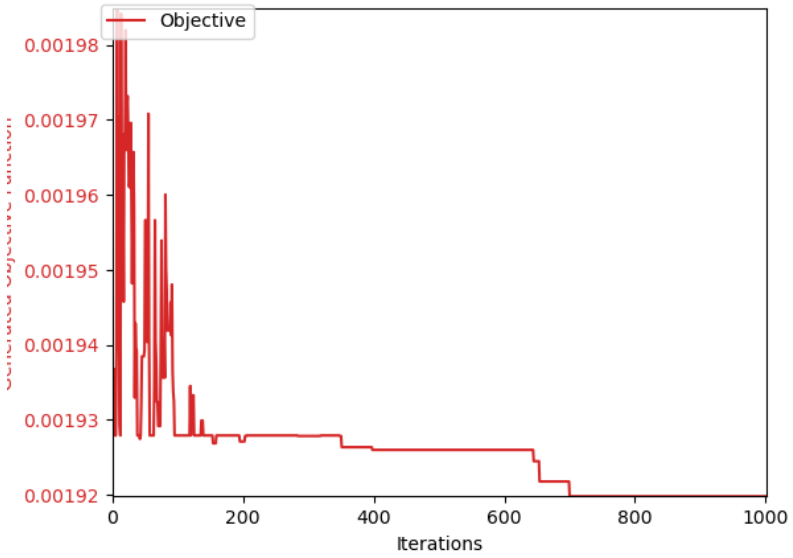

Figure 4.16: Current Fitness Over Iterations in Case Study 4

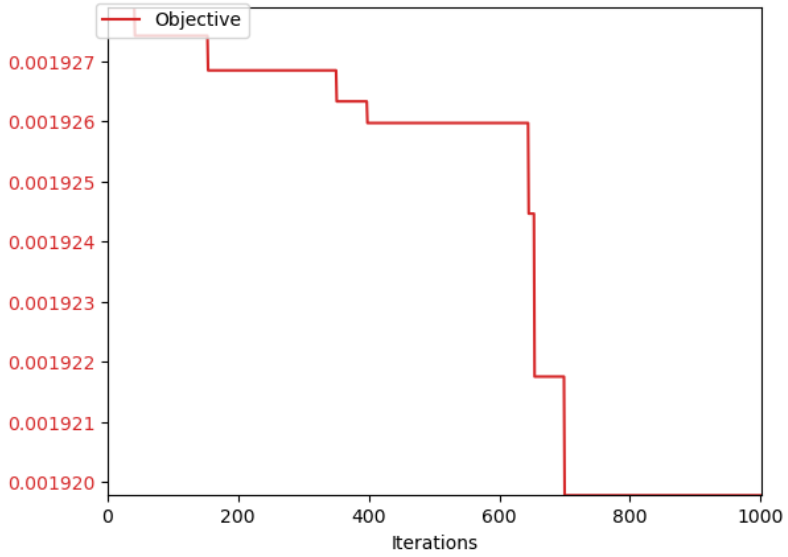

Figure 4.17: Best Fitness Over Iterations in Case Study 4

For this case, the difference was a bit bigger in the fitness as evident by the much simpler solution due to the geometric variant getting stuck in a local minima. We score 0.00191 as opposed to 0.00140 reported by Kunakote et al.'s study (2022) [2]. This time the ratio deviated a bit with 7 WTs producing 3kW giving 0.428kW/WT meaning a slightly worse efficiency and a bigger delta from the benchmark.

4.6 Case Study 5: 20x20 grid with geometric cooling schedule

For this test case, we use the geometric temperature decay function. The initial temperature is 1000, with 2 iterations per temperature step, a maximum of 500 iterations and a temperature reduction factor of 0.95. We model on a 20x20 grid, and the specified "dead cells" are included.

- **Grid Size:** 20x20
- **Dead Cells:** [(3,2),(4,2),(3,3),(4,3),(15,2),(16,2),(15,3),(16,3),(3,16),(4,16),(3,17), (4,17),(15,16),(16,16),(15,17),(16,17)]
- **Initial Temperature:** 1000
- **Final Temperature:** 0
- **Iterations per Temperature Step:** 2
- **Maximum Number of Iterations:** 500
- **Temperature Reduction Factor (α):** 0.95
- **Scheduling Function:** Geometric Temperature Decay Function

$$T = T * \alpha$$

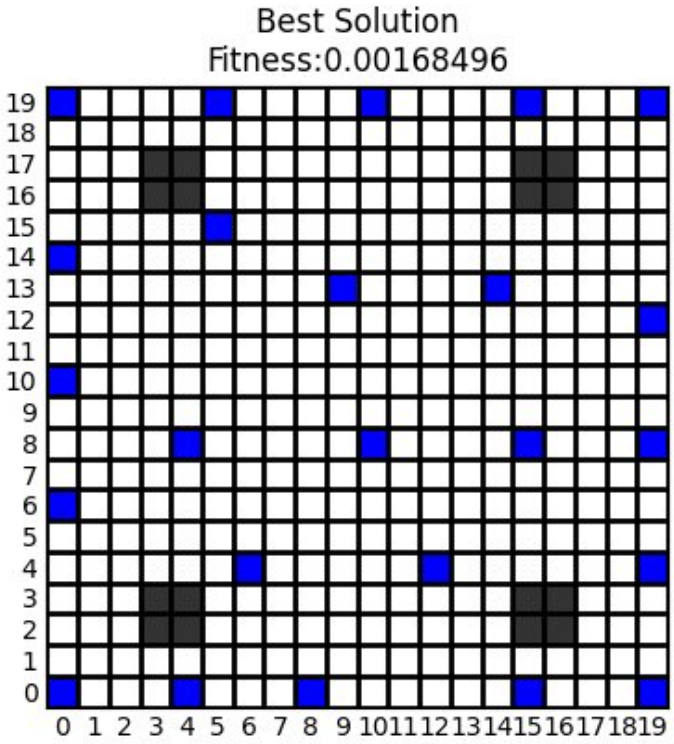

Figure 4.18: Best solution found by SA in case study 5

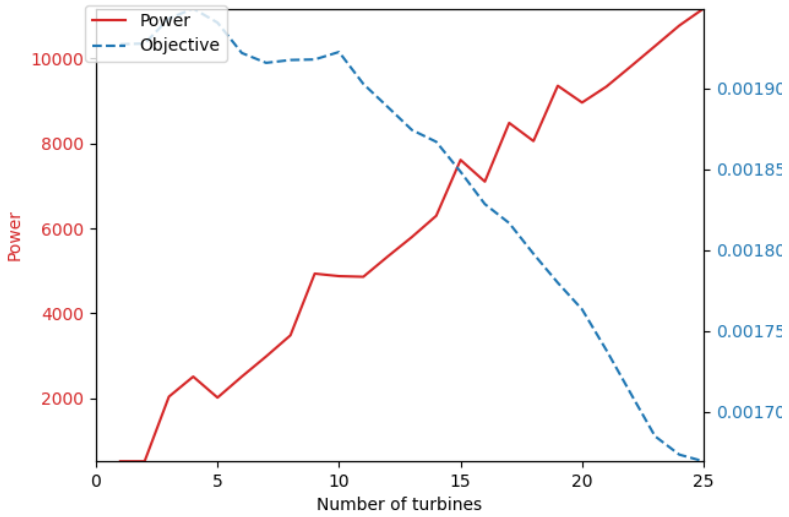

Figure 4.19: Comparison of Power Generation and Fitness Value with Number of Turbines in Case Study 5

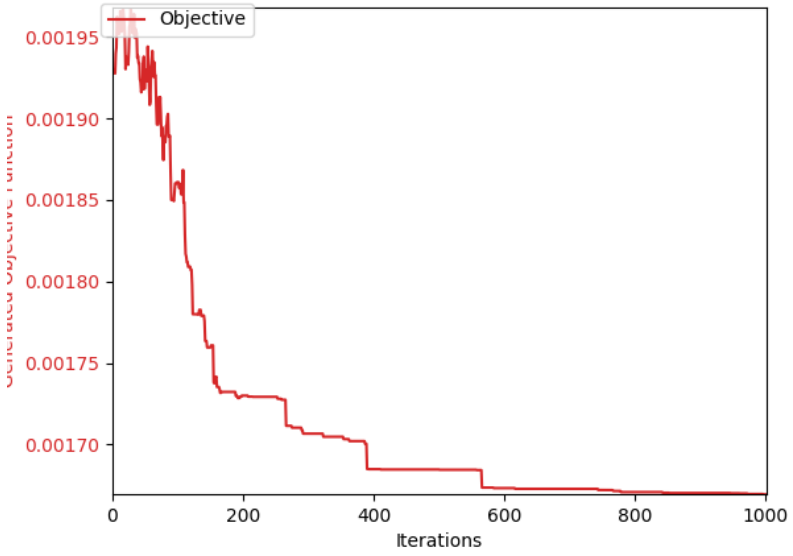

Figure 4.20: Current Fitness Over Iterations in Case Study 5

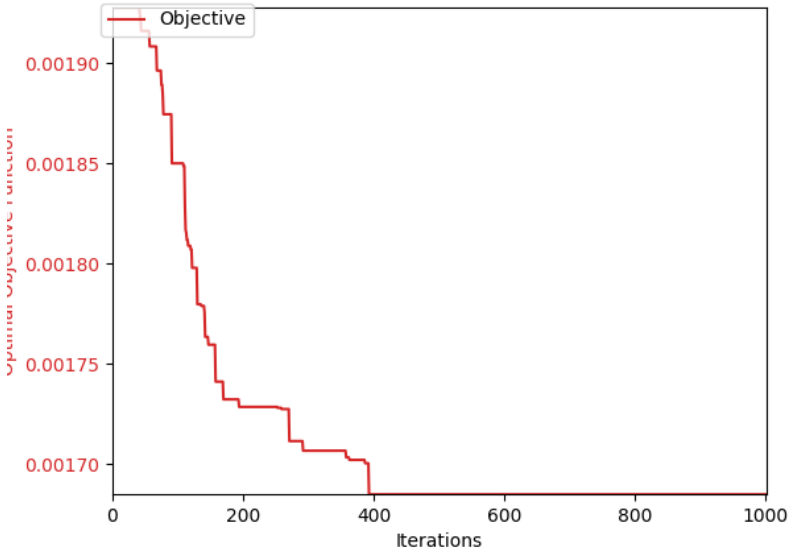

Figure 4.21: Best Fitness Over Iterations in Case Study 5

For this case, it produced nearly identical results as test case 2 for fitness and ratio of power/N.

4.7 Case Study 6: 25x25 grid with geometric cooling schedule

For the final test case, we use the geometric temperature decay function again. The initial temperature is 1000, with 2 iterations per temperature step, a maximum of 500 iterations and a temperature reduction factor of 0.95. We model on a 25x25 grid, and the specified "dead cells" are included.

- **Grid Size:** 25x25
- **Dead Cells:** [(5,5),(5,6),(6,5),(6,6),(5,18),(5,19),(6,18),(6,19),(18,5),(19,5),(18,6), (19,6),(18,18),(18,19), (19,18),(19,19),(7,7),(7,6),(7,5),(7,18),(7,19),(18,7),(19,7) ,(5,7), (6,7),(5,17),(6,17),(7,17),(17,5),(17,6),(17,7),(17,17),(17,18),(17,19),(18,17),(19,17)]
- **Initial Temperature:** 1000
- **Final Temperature:** 0
- **Iterations per Temperature Step:** 2
- **Maximum Number of Iterations:** 500
- **Temperature Reduction Factor (α):** 0.95
- **Scheduling Function:** Geometric Temperature Decay Function

$$T = T * \alpha$$

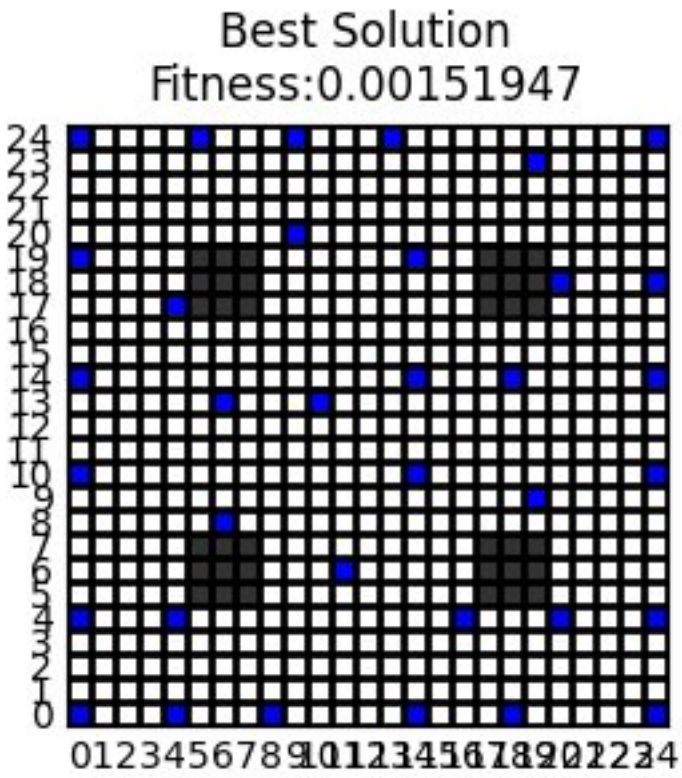

Figure 4.22: Best solution found by SA in case study 6

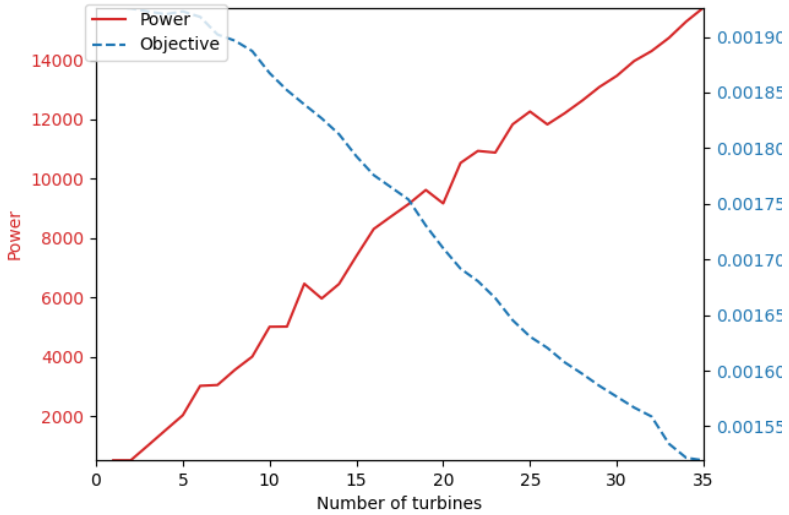

Figure 4.23: Comparison of Power Generation and Fitness Value with Number of Turbines in Case Study 6

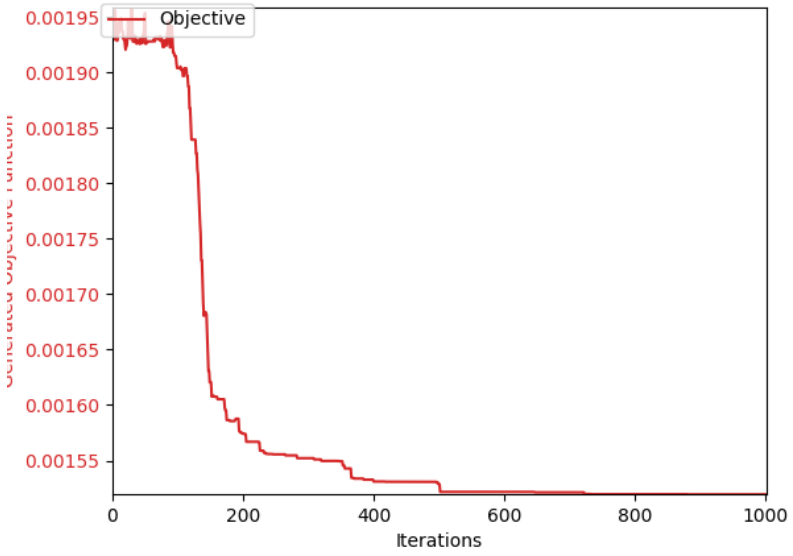

Figure 4.24: Current Fitness Over Iterations in Case Study 6

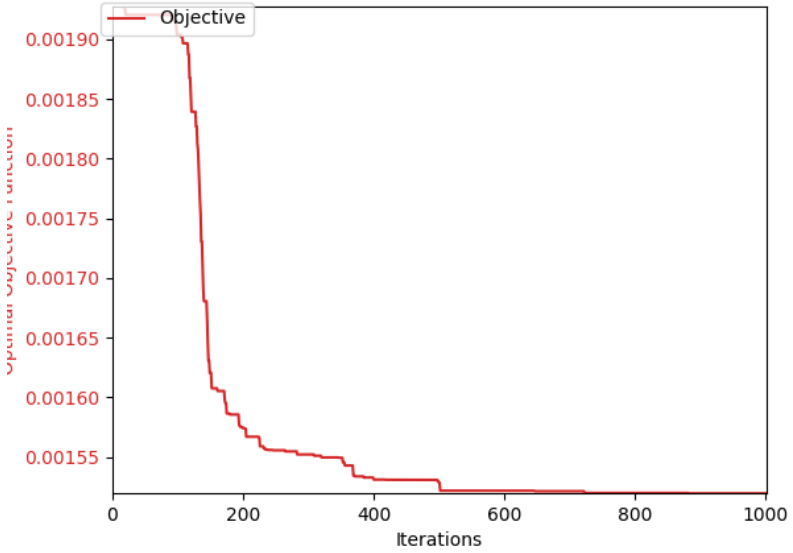

Figure 4.25: Best Fitness Over Iterations in Case Study 6

For this case, we achieved the lowest recorded fitness. We score 0.00151 as opposed to 0.00140 reported by Kunakote et al.'s study (2022) [2]. We had 35 WTs producing 16kW giving 0.457kW/WT which is again very close to the benchmark. It should be noted that while this case produced lower efficiency for the turbines, it is still more optimal as 4 less turbines were used greatly decreasing the cost and thus decreasing the cost/power.

4.8 Analysis of Wind farm Layout Optimization using Simulated Annealing

In the analysis of the wind farm layout optimization using simulated annealing, it is apparent that the choice of scheduling strategy, whether linear or geometric, significantly influences the convergence behavior and the quality of solutions achieved. The following observations were made based on the six test cases, encompassing grid sizes of 15×15, 20×20, and 25×25 for both linear and geometric scheduling:

Convergence Rate and Temperature Schedule Impact

The geometric scheduling strategy demonstrates a distinct characteristic in terms of its convergence behavior. As the optimization process unfolds, it becomes evident that geometric scheduling shows very little change in fitness values over iterations, particularly after the initial stages. (Compare Figure 7.3 Figure 7.5 Figure 7.7 with Figure 7.9 Figure 7.11 Figure 7.13) This stability is primarily due to the rapid decline in temperature, which reduces the probability of accepting new changes. Consequently, the algorithm becomes less inclined to explore the search space, limiting the opportunities for enhancing the solution.

Faster Achievement of Optimal Solutions with Geometric Scheduling

Because of the characteristic mentioned above, geometric scheduling tends to reach the best fitness value more quickly compared to linear scheduling. (Compare Figure 4.4 Figure 4.8 Figure 4.12 with Figure 4.16 Figure 4.20 Figure 4.24) This is because geometric simulated annealing focuses on fine-tuning solutions that it discovers early on and doesn't explore as much of the search space. However, this early convergence can also mean that it might miss out on potentially better solutions that lie further in the search space.

Illustrative Case: 15×15 Grid - Exploration vs. Early Convergence

A clear illustration of the trade-off between exploration and early convergence is provided by the 15×15 geometric test case. In this instance, the algorithm failed to discover the optimal solution that was successfully identified by the linear scheduling approach. It should be noted that in the 15×15 grid case, the optimal solutions tend to involve a higher number of wind turbines (WTs) since the cost decreases significantly as the number of WTs increases. The geometric scheduling method, however, converges prematurely, limiting its ability to explore the broader search space and ultimately selecting a solution with a lower number of WTs and a worse fitness value.

Convergence to 39 Wind Turbines in the 25×25 Grid

In the 25×25 linear test case, the algorithm selected a solution with 39 wind turbines (WTs), even though it explored configurations with more WTs. (notice Figure 4.11) This observation corresponds with existing research (Yang and Cho's research (2019) [1]), which suggests that there's an optimal configuration beyond which adding more turbines doesn't improve the objective.

In conclusion, the choice of scheduling strategy in simulated annealing for wind farm layout optimization has a profound impact on the convergence behavior and the quality of solutions attained. The geometric scheduling approach converges rapidly but may overlook superior solutions, while the linear scheduling method systematically explores the search space, potentially identifying more favorable configurations. This means that geometric cooling schedule is more favorable for problems with smaller search spaces where more fine tuning to the existing solution is required. However, as our problem had a much larger search space than conventional problems, linear scheduling performed generally better. Geometric scheduling fails to explore more and focuses only on exploitation leading to a higher chance of being trapped in a local minima.

Chapter 5

Genetic Algorithm

5.1 Algorithm

Genetic algorithms (GAs) are optimization algorithms inspired by the principles of natural selection and genetics. They operate by simulating the process of evolution to find solutions to complex problems. In a typical genetic algorithm, potential solutions to a problem are represented as individuals within a population. They, encoded as chromosomes, undergo selection based on their fitness, mimicking the survival of the fittest in nature. Crossover combines genetic information from selected individuals to create offspring, and mutation introduces random changes, maintaining genetic diversity. The fitness of these new individuals is evaluated, and over successive generations, the population evolves towards better solutions.

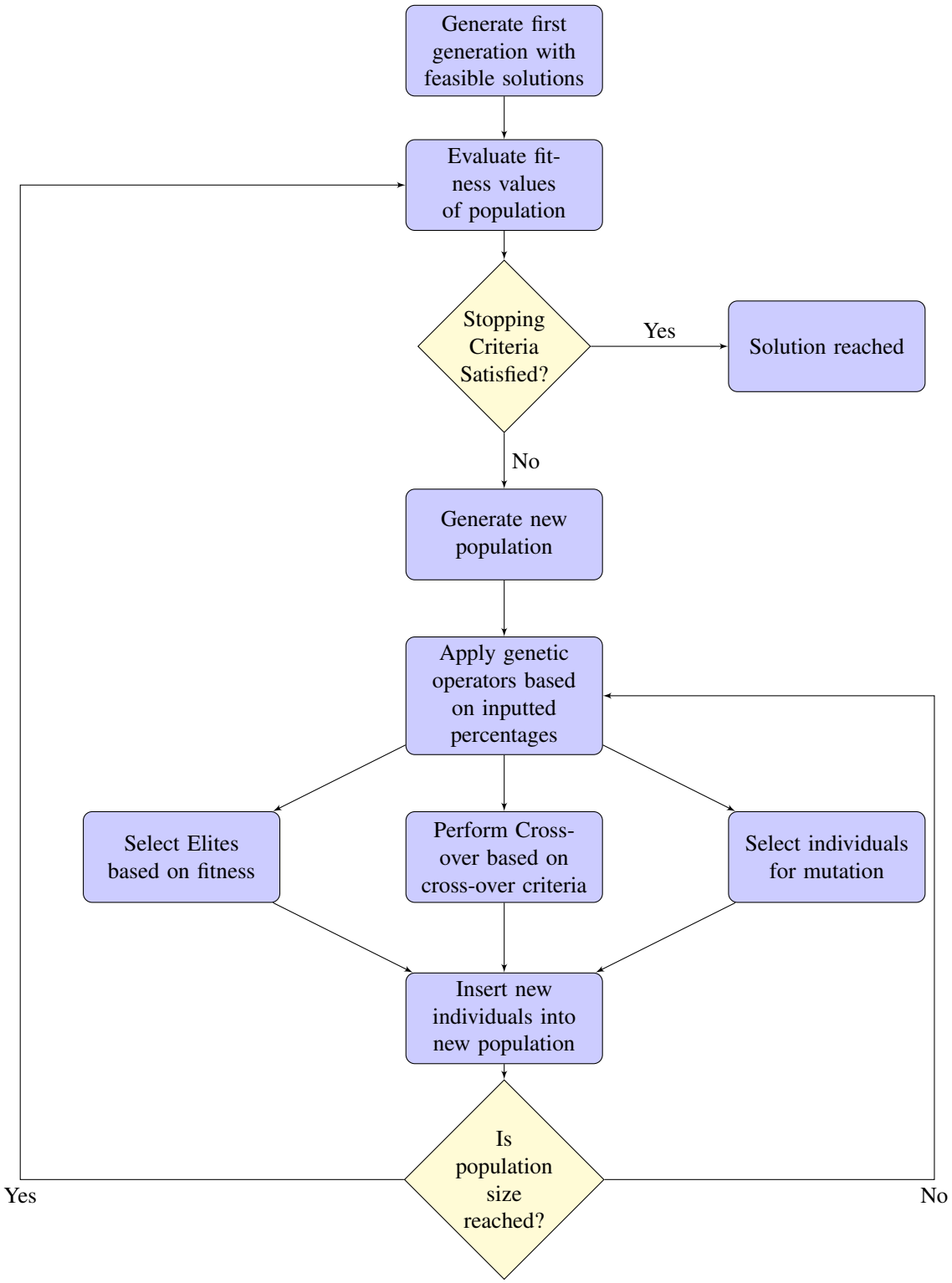

Figure 5.1: Flow chart of the genetic algorithm

5.1.1 Initialization:

1. **Setting Parameters:** The algorithm begins with a specified population size and creates generations until convergence or an artifical threshold. It creates the solution based on elite, crossover, and mutation percentages.
2. **Initial Population:** Generate an initial random population of potential solutions (individuals). Each individual (chromosome) represents a possible solution to the problem.

5.1.2 Iterative Process

1. **Evaluation:** Assess the fitness of each chromosome and then sort the population by fitness.
2. **Elitism:** Select a percentage of the fittest individuals from the current population to be taken as persisting elite members.
3. **Crossover:** Pair selected individuals and create offspring by combining their genetic information. Selection occurs by rank selection where more fit members are more likely to be chosen according to the crossover percentage. The algorithm randomly selects from 2 possible ways to perform crossover:
    - Uniform Crossover where a random number of genes are selected to be swapped, while fixing the two resulting children ensuring they satisfy all constraints by perturbing each violating gene in the neighbouring cells until a configuration is found.
    - One Point Crossover where a cutoff line is made in the grid and all turbines after the line are swapped across the parents. The same constraint fixing is done on the resulting children.
4. **Mutation:** is performed to introduce small random changes to some a number of chromosomes in the population depending on the mutation percentage. To perform mutation, we simulate bit flipping on a random number of genes by either removing, adding (Wind turbine added in the position), or moving a turbine to a new position with constraint fixing. Through experimenting, we found that it is better to not mutate a random amount of genes and only mutate 1 to ensure the algorithm converges to a solution.
5. **Replacement:** Replace the old population with the new one (composed of parents and offspring).
6. **Tracking the Best Solution:** Throughout its run, the algorithm keeps track of the best solution encountered so far, ensuring that this information is preserved even if subsequent generations produce inferior results.
7. **Termination:** The algorithm terminates when all solutions converge to the same value

or when an artificial max number of generations is imposed and reached.

5.1.3 Conclusion

Through the iterative process of generating, evaluating, and evolving potential solutions, genetic algorithms emulate the principles of natural selection to discover optimal configurations. Unlike traditional optimization methods, genetic algorithms embrace the diversity of the solution space by maintaining a population of potential solutions. This diversity, coupled with the incorporation of genetic operators such as crossover and mutation, allows the algorithm to explore a broad range of possibilities. By favoring the fittest individuals in each generation, genetic algorithms systematically evolve towards better solutions. This adaptability and the ability to escape local optima make genetic algorithms well-suited for tackling intricate problems, where the optimal solution may not be immediately apparent within a large search space.

5.2 Case Study 1: 15x15 grid with 2 Crossover methods

- **Grid Size:** 15x15
- **Dead Cells:** [(2,2), (12,2), (2,12), (12,12)]
- **Population Size:** 50
- **Maximum Number of Generations:** 300
- **Survivor Percentage:** 10%
- **Cross-Over Percentage:** 80%
- **Mutation Percentage:** 10%
- **Cross-Over Criteria:** Randomly selected between One Point Cross-over and Uniform Cross-over.

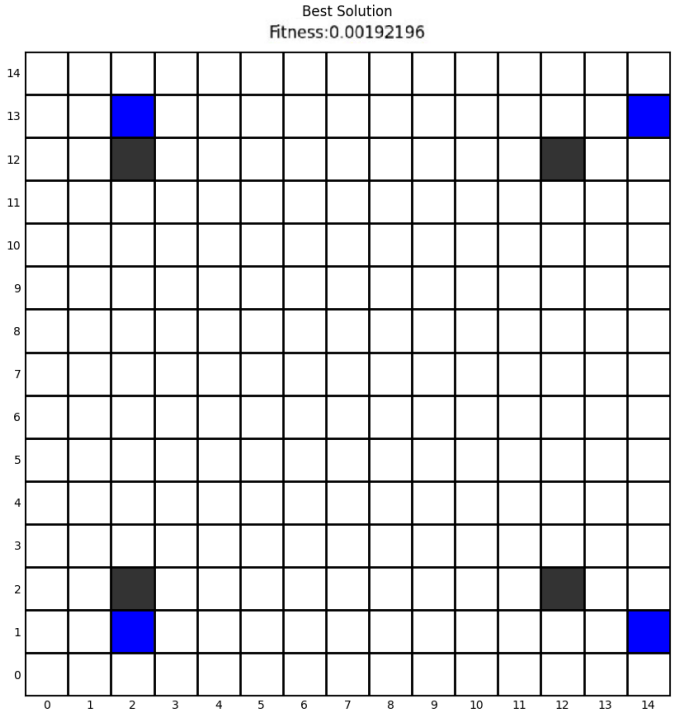

Figure 5.2: Best solution found by GA in case study 1

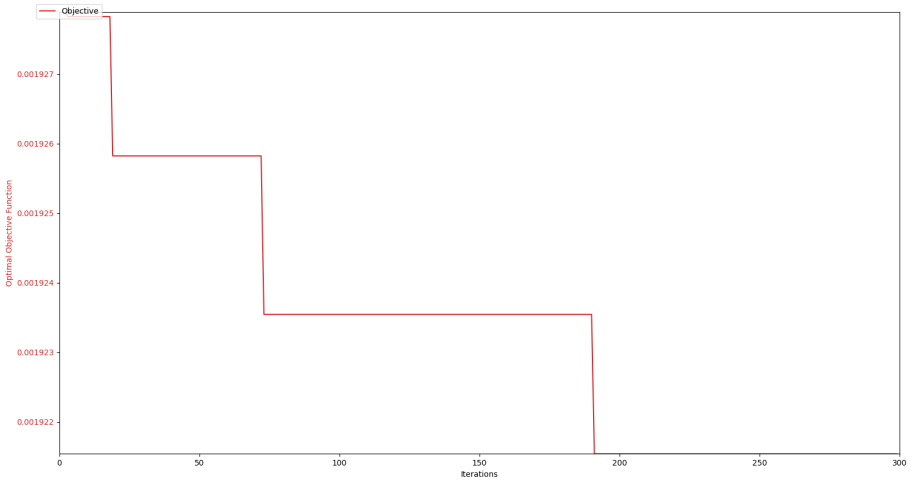

Figure 5.3: Best Fitness Over Generations in Case Study 1

When comparing the value of the best solution to known benchmarks, our implementation scored 0.00192 while Hassoine, Laglou, Addaim, and Madi (2019) [7] scored 0.00149 which is

much better than our solution. This is clear to see as the solution got stuck in a local minima of only placing 4 turbines. This was similar to what occurred in the geometric case of simulated annealing. This problem size in specific seems to have a much smaller jumps in fitness values for each added wind turbine making it harder for the algorithm to search in the direction of adding more turbines leading to this entrapment in the minima. This may also be an artefact of limiting the mutation for just 1 gene however this was necessary for bigger problem instances.

5.3 Case Study 2: 20x20 grid with 2 Crossover methods

- **Grid Size:** 20x20
- **Dead Cells:** [(3,2),(4,2),(3,3),(4,3),(15,2),(16,2),(15,3),(16,3),(3,16),(4,16),(3,17),(4,17),(15,16),(16,16),(15,17),(16,17)]
- **Population Size:** 50
- **Maximum Number of Generations:** 300
- **Survivor Percentage:** 10%
- **Cross-Over Percentage:** 80%
- **Mutation Percentage:** 10%
- **Cross-Over Criteria:** Randomly selected between One Point Cross-over and Uniform Cross-over.

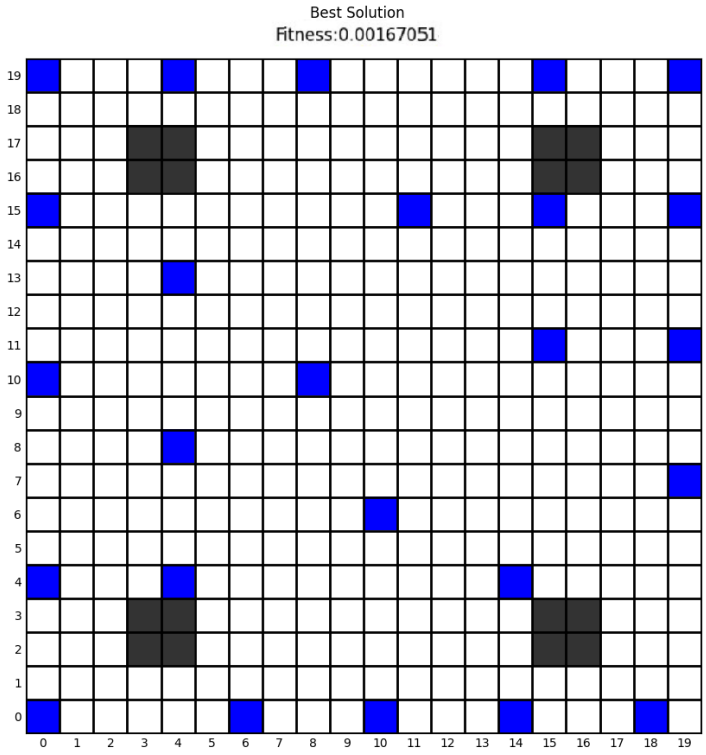

Figure 5.4: Best solution found by GA in case study 2

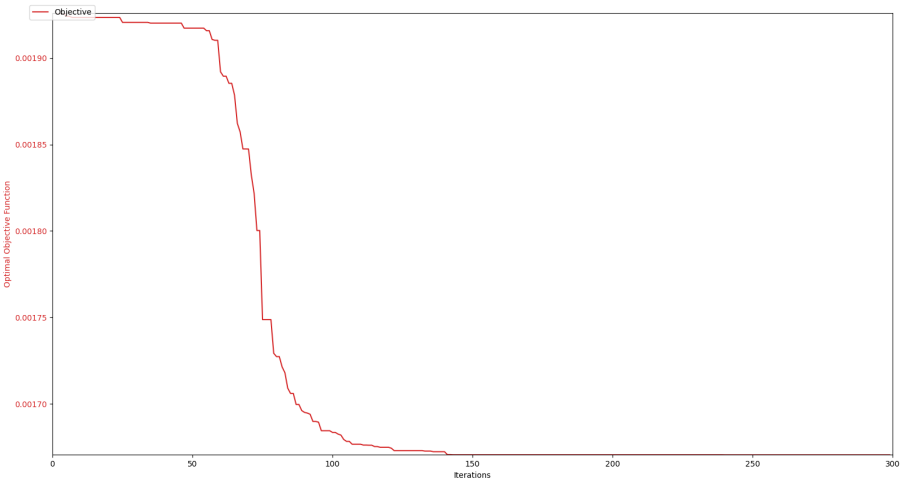

Figure 5.5: Best Fitness Over Generations in Case Study 2

For this case, the difference became smaller in the fitness. We scored 0.00167 as opposed to

0.00149 of the benchmark. We also overtook the performance of simulated annealing both in the linear and geometric cases giving the overall lowest fitness for the 20x20 problem.

5.4 Case Study 3: 25x25 grid with 2 Crossover methods

- **Grid Size:** 25x25
- **Dead Cells:** [(5,5),(5,6),(6,5),(6,6),(5,18),(5,19),(6,18),(6,19),(18,5),(19,5),(18,6) ,(19,6),(18,18),(18,19), (19,18),(19,19),(7,7),(7,6),(7,5),(7,18),(7,19),(18,7),(19,7) ,(5,7), (6,7),(5,17),(6,17),(7,17),(17,5),(17,6),(17,7),(17,17),(17,18),(17,19),(18,17),(19,17)]
- **Population Size:** 50
- **Maximum Number of Generations:** 300
- **Survivor Percentage:** 10%
- **Cross-Over Percentage:** 80%
- **Mutation Percentage:** 10%
- **Cross-Over Criteria:** Randomly selected between One Point Cross-over and Uniform Cross-over.

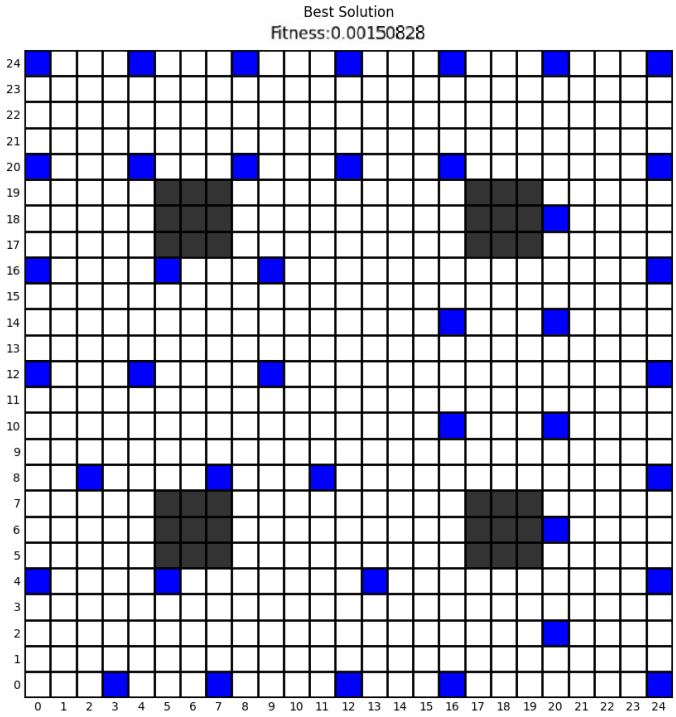

Figure 5.6: Best solution found by GA in case study 3

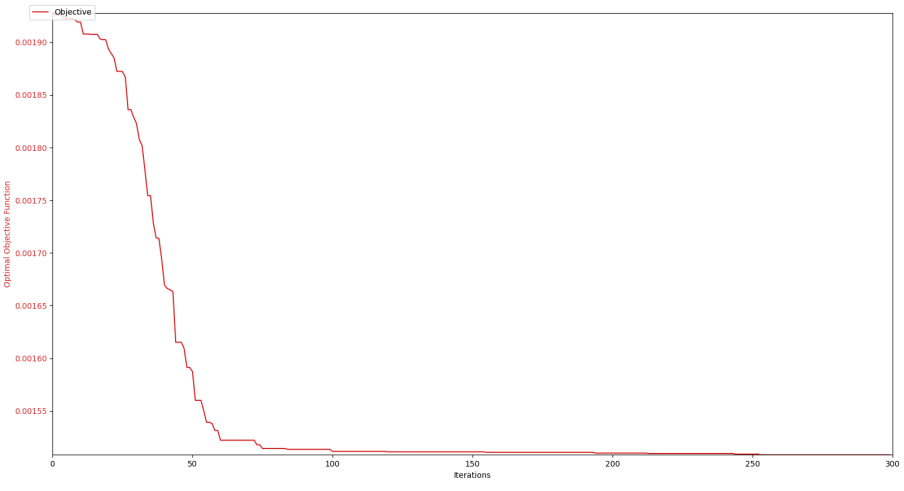

Figure 5.7: Best Fitness Over Generations in Case Study 3

For this case, the difference became even smaller in the fitness. We score 0.00150 which is very close to the benchmark giving our lowest fitness achieved yet. We also surpass the

annealing values in both cases settling this as our new benchmark so far.

5.5 Case Study 4: 15x15 grid with 1 Crossover Method

- **Grid Size:** 15x15
- **Dead Cells:** [(2,2), (12,2), (2,12), (12,12)]
- **Population Size:** 50
- **Maximum Number of Generations:** 300
- **Survivor Percentage:** 10%
- **Cross-Over Percentage:** 80%
- **Mutation Percentage:** 10%
- **Cross-Over Criteria:** One Point Cross-over.

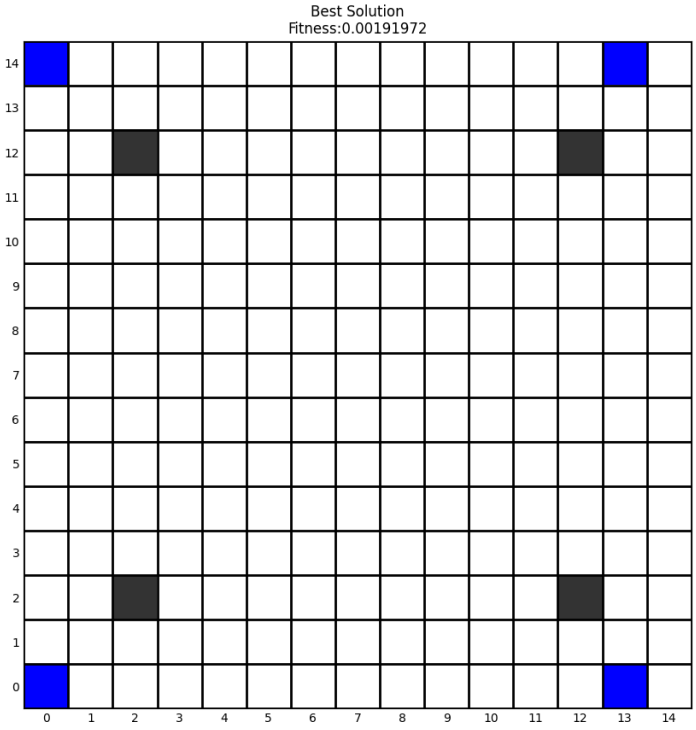

Figure 5.8: Best solution found by GA in case study 4

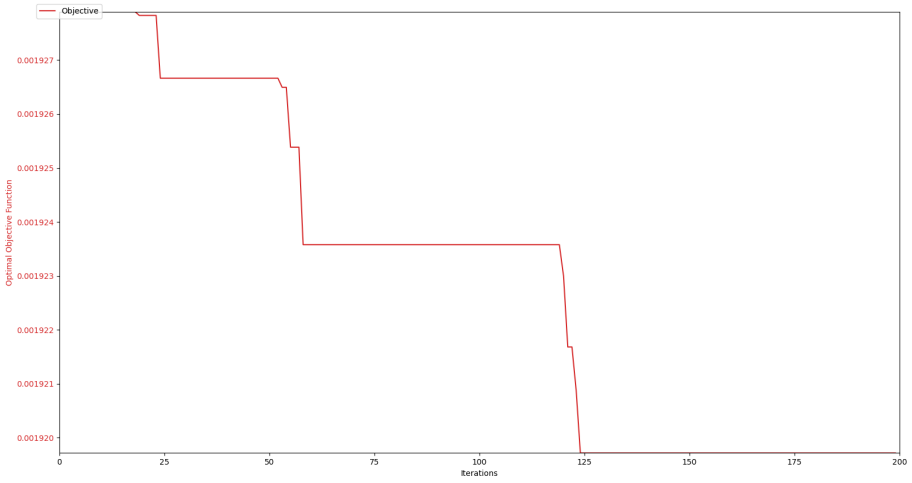

Figure 5.9: Best Fitness Over Generations in Case Study 4

For this case, we surprisingly get a better result than case 1. This is unexpected as using

just 1 form of crossover makes the exploration of the algorithm overall weaker. This could be attributed to the stochastic nature of the algorithms, and we will elaborate further on this later in the statistical analysis.

5.6 Case Study 5: 20x20 grid with 1 Crossover Method

- **Grid Size:** 20x20
- **Dead Cells:** [(3,2),(4,2),(3,3),(4,3),(15,2),(16,2),(15,3),(16,3),(3,16),(4,16),(3,17),(4,17),(15,16),(16,16),(15,17),(16,17)]
- **Population Size:** 50
- **Maximum Number of Generations:** 300
- **Survivor Percentage:** 10%
- **Cross-Over Percentage:** 80%
- **Mutation Percentage:** 10%
- **Cross-Over Criteria:** One Point Cross-over.

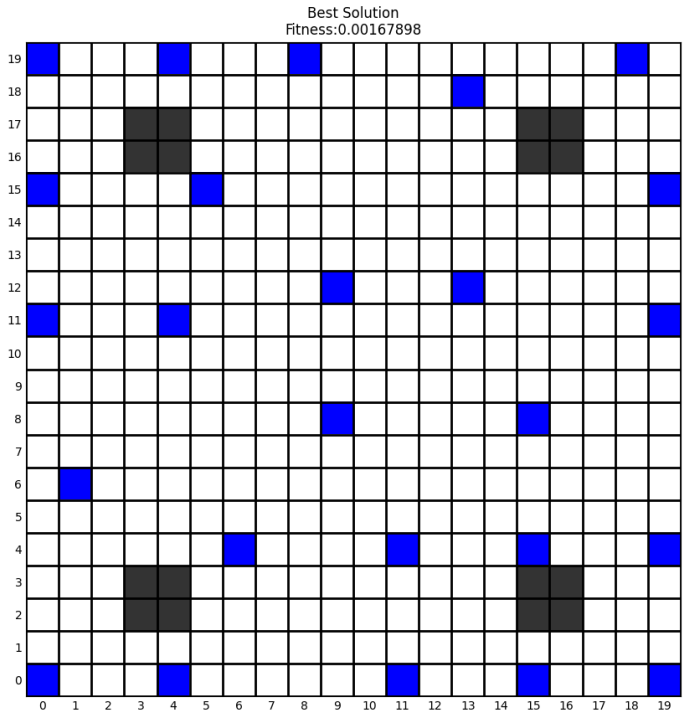

Figure 5.10: Best solution found by GA in case study 5

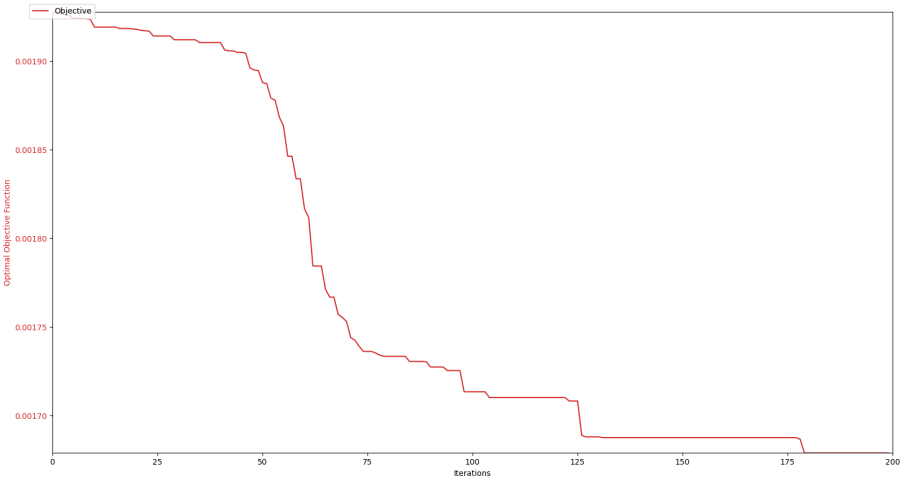

Figure 5.11: Best Fitness Over Generations in Case Study 5

For this case, it produced nearly identical results as test case 2 for fitness with a very slight drop.

5.7 Case Study 6: 25x25 grid with 1 Crossover Method

- **Grid Size:** 25x25
- **Dead Cells:** [(5,5),(5,6),(6,5),(6,6),(5,18),(5,19),(6,18),(6,19),(18,5),(19,5),(18,6), (19,6),(18,18),(18,19), (19,18),(19,19),(7,7),(7,6),(7,5),(7,18),(7,19),(18,7),(19,7) ,(5,7), (6,7),(5,17),(6,17),(7,17),(17,5),(17,6),(17,7),(17,17),(17,18),(17,19),(18,17),(19,17)]
- **Population Size:** 50
- **Maximum Number of Generations:** 300
- **Survivor Percentage:** 10%
- **Cross-Over Percentage:** 80%
- **Mutation Percentage:** 10%
- **Cross-Over Criteria:** One Point Cross-over.

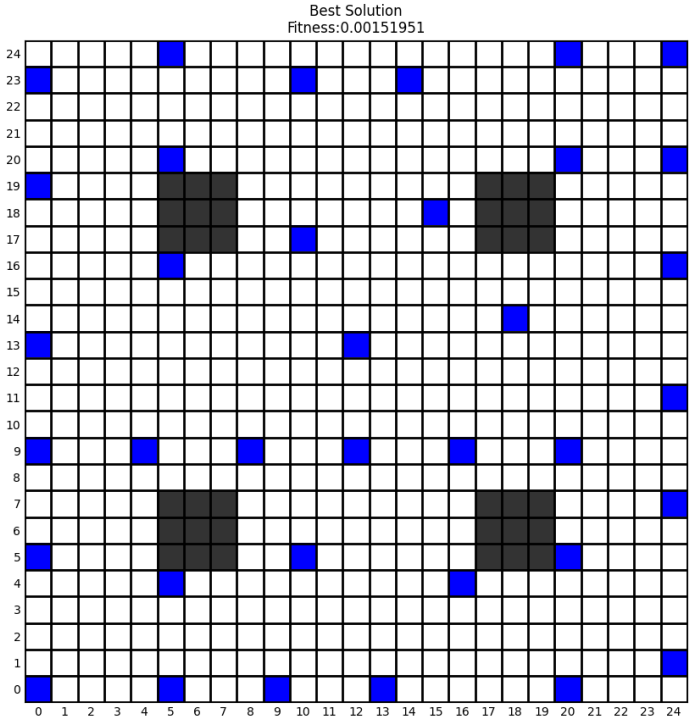

Figure 5.12: Best solution found by GA in case study 6

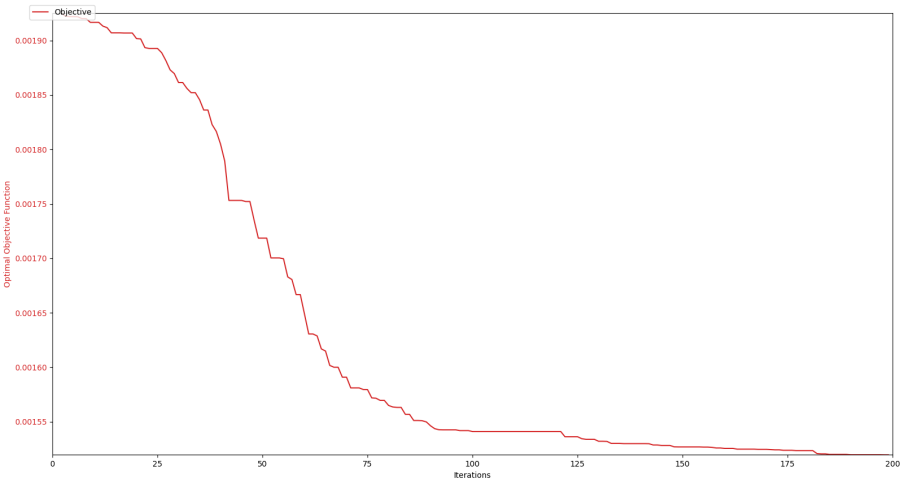

Figure 5.13: Best Fitness Over Generations in Case Study 6

For this case, we get a worse result than case 3 and this is the first time we see the gap widen

between the configurations. Using only 1 type of crossover leads to lower exploration leading to finding slightly worse solution and we predict that as the problem size increases further, this gap will widen even more. Moreover we still retain a close fitness to the benchmark.

5.8 Analysis of Wind farm Layout Optimization using Genetic Algorithm

In the analysis of the wind farm layout optimization using genetic algorithm, it is apparent that the choice of crossover influences the quality of solutions achieved. The following observations were made based on the six test cases, encompassing grid sizes of 15×15, 20×20, and 25×25 for both configurations:

Crossover Strategy Impact

We observe that the choice of crossover strategy influences the quality of the solutions produced. For a choice of just one point crossover, not enough randomness exists for the algorithm to effectively explore the entire solution space leading to overall worse solutions. The algorithm also takes much longer to converge with just one point crossover due to the lack of uniform crossover which manages to retain more of the parent in each crossover operation leading to quicker convergence. Thus using a combination of the two strikes the best balance between exploration and exploitation

Illustrative Case: 20×20 Grid - Convergence

A clear illustration of the differences in crossover strategies is provided by the 20×20 cases. In this instance, the single crossover variant not only reached a worse solution, but also took longer to converge to that solution as can be seen in Figures Figure 7.6 and Figure 7.12.

Search Space Coverage

We also notice a trend in all the fitness values obtained in our testing compared to simulated annealing. It consistently outperforms the annealing algorithm for every problem size due to the much stronger capabilities of the genetic algorithm and population based algorithms in general to search the space more effectively and converge on a global minima. These results showcase the power of population based algorithms in not just quality of solutions, but also speed of convergence.

Illustrative Case: 20×20 Grid - Space Coverage

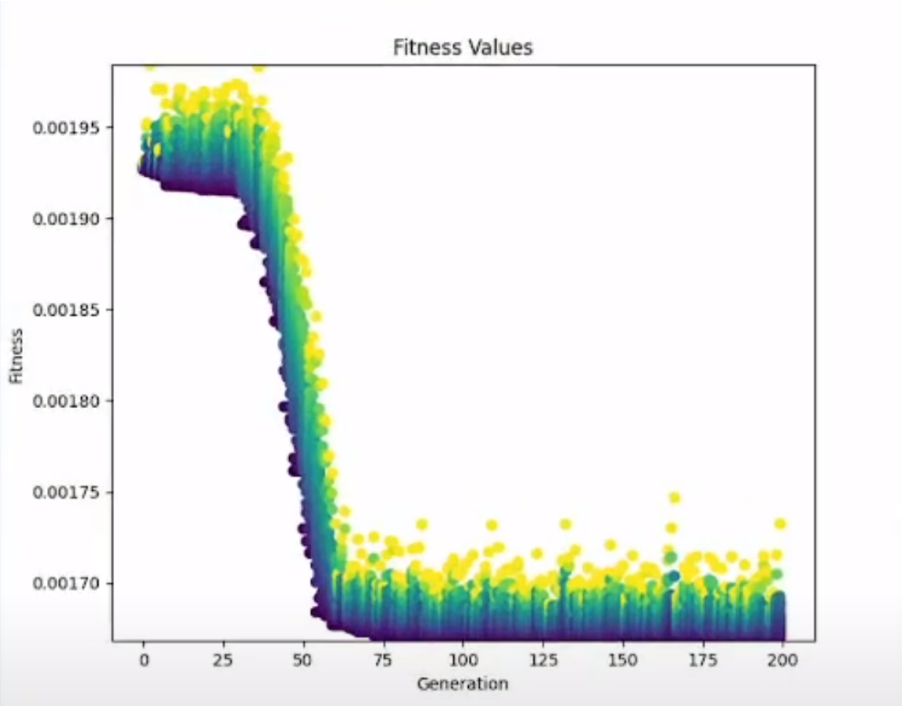

Figure 5.14: Genetic algorithm population fitnesses against generations

As can be seen for the 20×20 case, the genetic algorithm thoroughly explores the search space by considering multiple solutions and iteratively augmenting them through each generation. It can be seen that there are many outliers (mutants in yellow) and these are what allow the GA to explore better than just the single agent present in simulated annealing. As the iterations pass, the GA starts converging towards the best solution discovered yet (elites in purple).

Chapter 6

Particle Swarm Optimization

6.1 Algorithm

Particle Swarm Optimization (PSO) is a population-based optimization algorithm inspired by the social behavior of birds. In this approach, potential solutions to a problem are abstracted as particles in a multidimensional search space. These particles adjust their positions iteratively based on their own historical best positions and the collective knowledge of the swarm's best-known position. The algorithm employs a velocity update mechanism that combines inertia, cognitive (personal best), and social (global best) components to guide the particles toward optimal solutions. This iterative process enables PSO to effectively explore the solution space in the early stages, transitioning to exploitation of promising regions as the swarm converges.

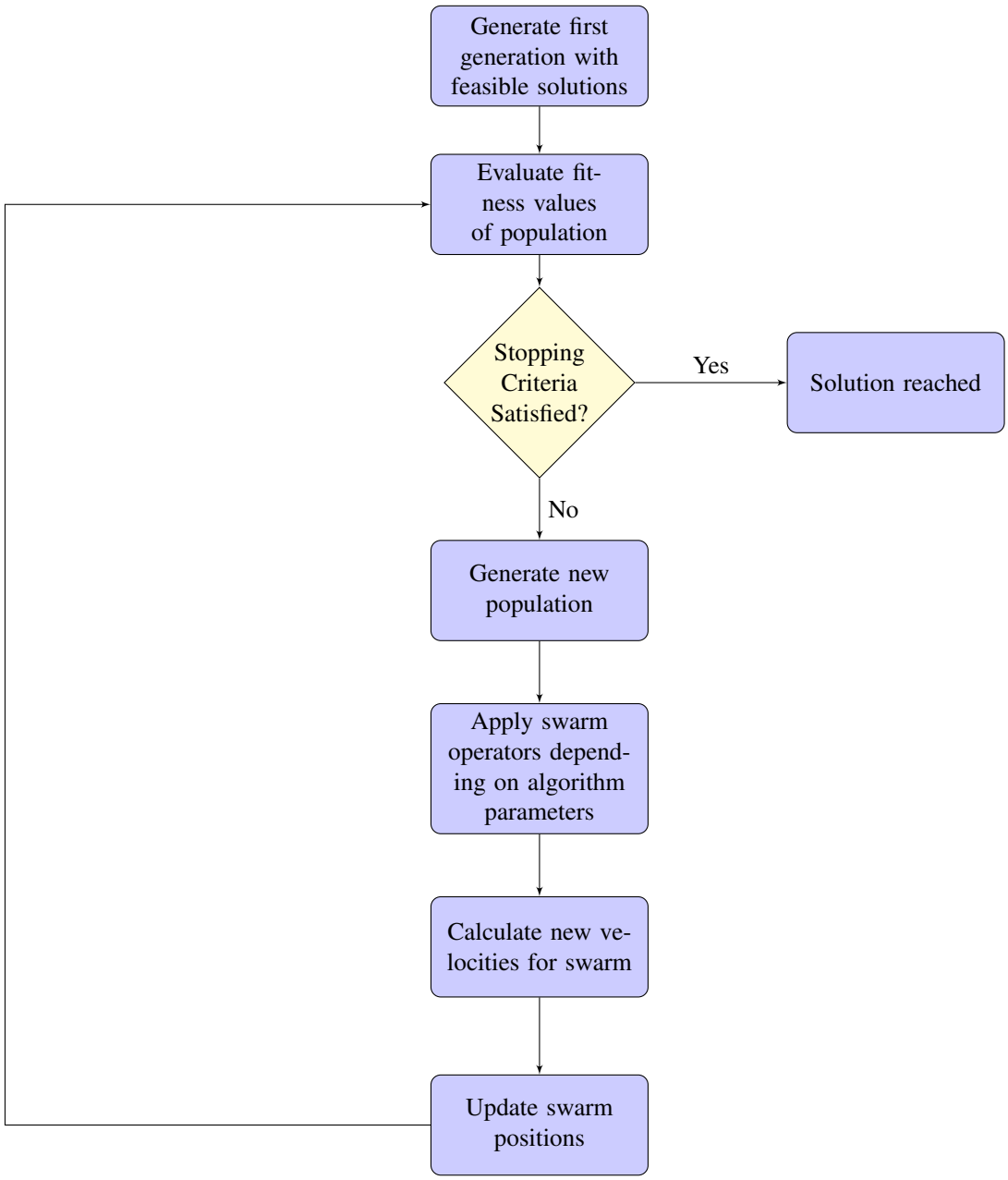

Figure 6.1: Flow chart of the particle swarm optimization algorithm

6.1.1 Initialization:

1. **Setting Parameters:** The algorithm begins by defining parameters such as the population size, inertia weight, cognitive and social coefficients. These parameters influence the behavior of the particles and are crucial for the convergence of the algorithm.
2. **Initial Population:** Initialize a swarm of particles, each representing a potential solution to the problem. The position of each particle corresponds to a possible solution, and the velocity represents the direction and magnitude of movement in the solution space.

6.1.2 Iterative Process

1. **Evaluation:** Assess the fitness of each particle based on the objective function. The fitness values guide the particles in their search for optimal solutions.
2. **Update Personal Best:** Each particle maintains its personal best position based on its own historical performance, considering the best solution it has encountered so far.
3. **Update Global Best:** Identify the global best position among multiple particles. This position represents the overall best solution found by any particle in its cluster.
4. **Particle Movement:** Update the velocity and position of each particle using a combination of its current velocity, personal best position, and the global best position. This movement is influenced by inertia, individual cognitive factors, and collective social factors:

$$\mathbf{v}_i^{t+1} = w \cdot \mathbf{v}_i^t + c_1 \cdot r_1 \cdot (\mathbf{p}_i^t - \mathbf{x}_i^t) + c_2 \cdot r_2 \cdot (\mathbf{g}^t - \mathbf{x}_i^t) \tag{6.1}$$

where for particle i, v_i^{t+1} is its new velocity at step $t+1$, x_i^{t+1} is its position at step t, p_i^t is its personal best position at step t, g_i^t is the global best position at step t, w is the weight of inertia, $c1$ and $c2$ are cognitive and social factors, $r1$ and $r2$ are 2 random numbers from 0 to 1.

Since PSO operates in a continuous space, we use a custom made discrete PSO by considering each wind turbine position as a coordinate and approximating to the nearest cell. This allows the algorithm to perturb the position of each turbine in a continuous space to get closer to the optima without sacrificing any precision.

For a solution to get close to another solution, a two-step process is implemented. First, the existing turbines in the first solution need to approach their respective counterparts in the second solution. Second, it is essential for the number of turbines to be identical in both solutions. In this case, the number of wind turbines is treated as an additional variable in each solution in addition to its wind turbine positions. To implement this, the

velocity vector is defined to include vectors for moving turbines and special operators designed to add or remove turbines at specific positions, or randomly over time.

The update occurs using custom defined operators [8] for addition, subtraction, and multiplication:

- **Addition**: Performs element wise addition for each turbine position in the solution. Special (add or remove) operators are appended with no modification.
- **Subtraction**: Performs element wise subtraction for each turbine position in the solution. Trailing turbines in the first solution that don't have corresponding turbines are appended as add special operators. Trailing turbines in the second solution that don't have corresponding turbines are appended as remove special operators.
- **Multiplication**: Multiplies the entire solution element wise by a scalar. If the scalar is less than 1, a random percentage of the turbines is removed based on the scalar value. If it's more than 1, a random percentage of the turbines is added based on the scalar value.

5. **Constraint Handling:** Ensure that the updated positions of particles adhere to any problem-specific constraints. If necessary, adjust the particle's position to satisfy the constraints.
6. **Tracking the Best Solution:** Keep track of the best solution encountered throughout the iterations to preserve information about the optimal configuration.
7. **Termination:** The algorithm terminates after a specified number of iterations.

6.1.3 Conclusion

In the iterative process of Particle Swarm Optimization (PSO), a swarm of particles collaboratively explores the solution space to find optimal configurations. The particles adjust their positions and velocities based on personal experiences, as well as information gleaned from the entire swarm. The balance between exploration and exploitation is governed by parameters like inertia weight and cognitive and social coefficients. By updating personal best positions and sharing information about global best solutions, PSO leverages the collective intelligence of the swarm to converge towards optimal solutions.

6.2 Case Study 1: 15x15 grid with Ring Topology

- **Grid Size:** 15x15
- **Dead Cells:** [(2,2), (12,2), (2,12), (12,12)]

- **Population Size:** 50
- **Maximum Number of Generations:** 100
- **Inertia Weight:** 0.792
- **Cognitive Learning Factor:** 1.4944
- **Social Learning Factor:** 1.4944
- **Topology:** Ring with 2 neighbours

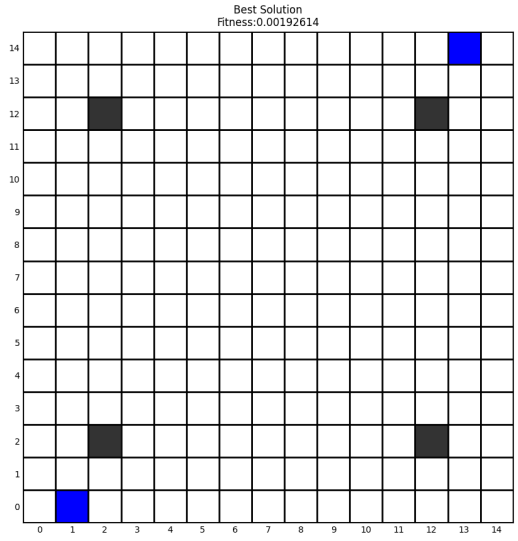

Figure 6.2: Best solution found by PSO in case study 1

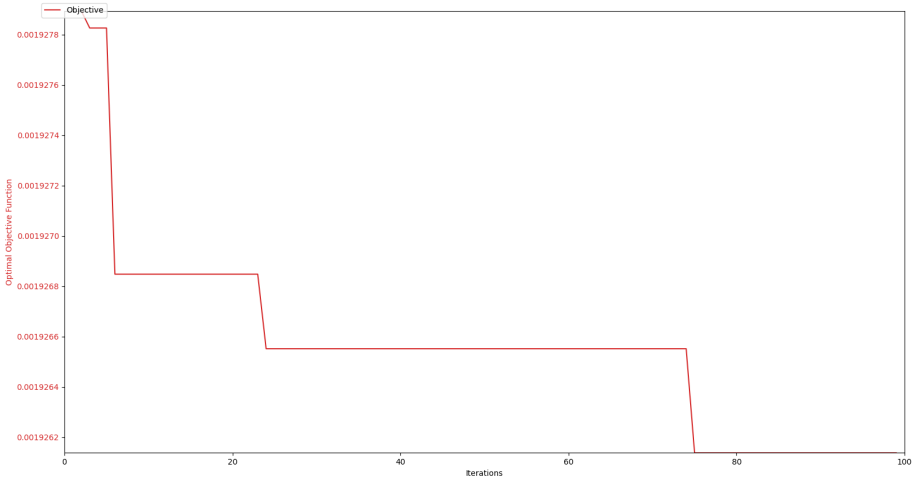

Figure 6.3: Best Fitness Over Generations in Case Study 1

When compared to the value of GA, our implementation scored 0.001926 while the better variant of GA scored 0.001919 which is quite close. This algorithm when coupled with ring topology also fails to escape the local minima of the low number of turbines.

6.3 Case Study 2: 20x20 grid with Ring Topology

- **Grid Size:** 20x20
- **Dead Cells:** [(3,2),(4,2),(3,3),(4,3),(15,2),(16,2),(15,3),(16,3),(3,16),(4,16),(3,17),(4,17),(15,16),(16,16),(15,17),(16,17)]
- **Population Size:** 50
- **Maximum Number of Generations:** 100
- **Inertia Weight:** 0.792
- **Cognitive Learning Factor:** 1.4944
- **Social Learning Factor:** 1.4944
- **Topology:** Ring with 2 neighbours

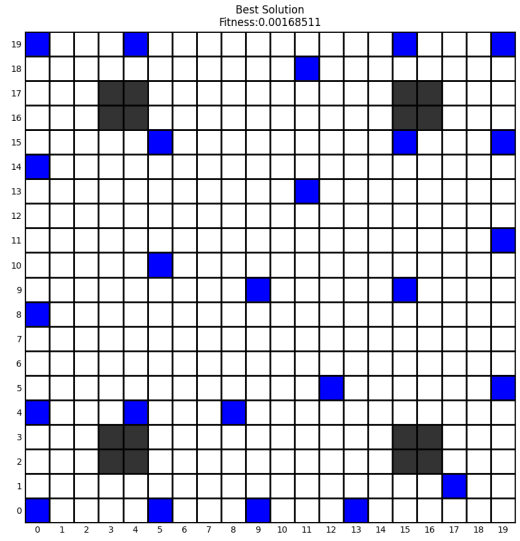

Figure 6.4: Best solution found by PSO in case study 2

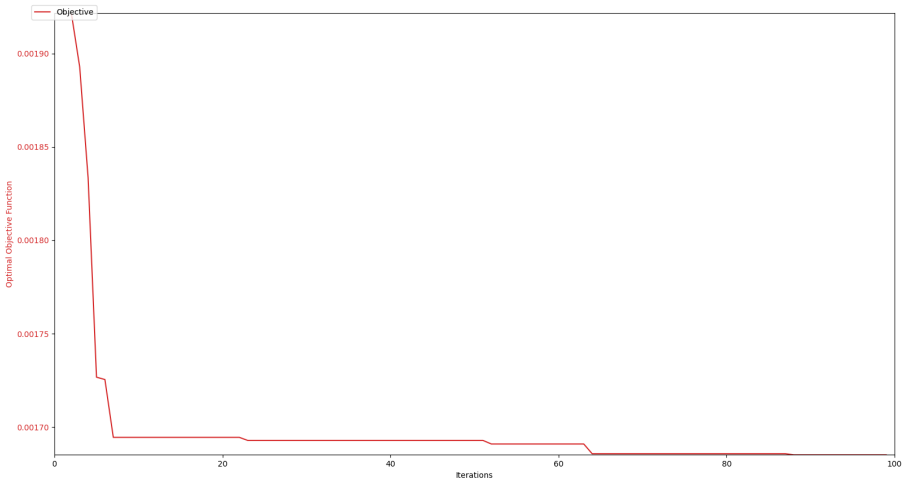

Figure 6.5: Best Fitness Over Generations in Case Study 2

For this case, the difference became smaller in the fitness. We scored close to GA with 0.00168 as opposed to 0.00167. We also overtook the performance of simulated annealing both in the linear and geometric cases reaching a good middle ground between SA and GA. The main advantage over GA is the much faster execution times as will become evident in our analysis section.

6.4 Case Study 3: 25x25 grid with Ring Topology

- **Grid Size:** 25x25
- **Dead Cells:** [(5,5),(5,6),(6,5),(6,6),(5,18),(5,19),(6,18),(6,19),(18,5),(19,5),(18,6) ,(19,6),(18,18),(18,19), (19,18),(19,19),(7,7),(7,6),(7,5),(7,18),(7,19),(18,7),(19,7) ,(5,7), (6,7),(5,17),(6,17),(7,17),(17,5),(17,6),(17,7),(17,17),(17,18),(17,19),(18,17),(19,17)]
- **Population Size:** 50
- **Maximum Number of Generations:** 100
- **Inertia Weight:** 0.792
- **Cognitive Learning Factor:** 1.4944
- **Social Learning Factor:** 1.4944
- **Topology:** Ring with 2 neighbours

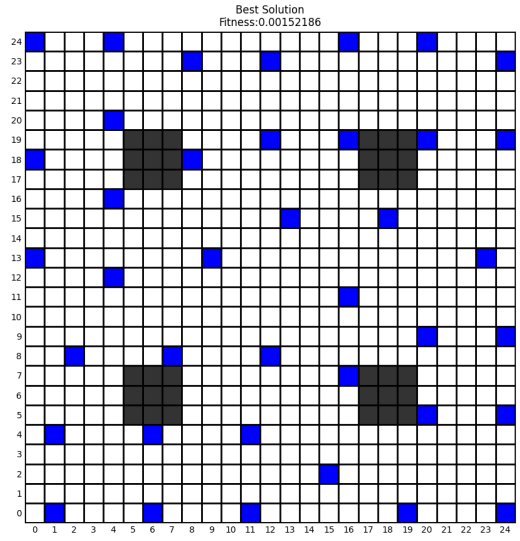

Figure 6.6: Best solution found by PSO in case study 3

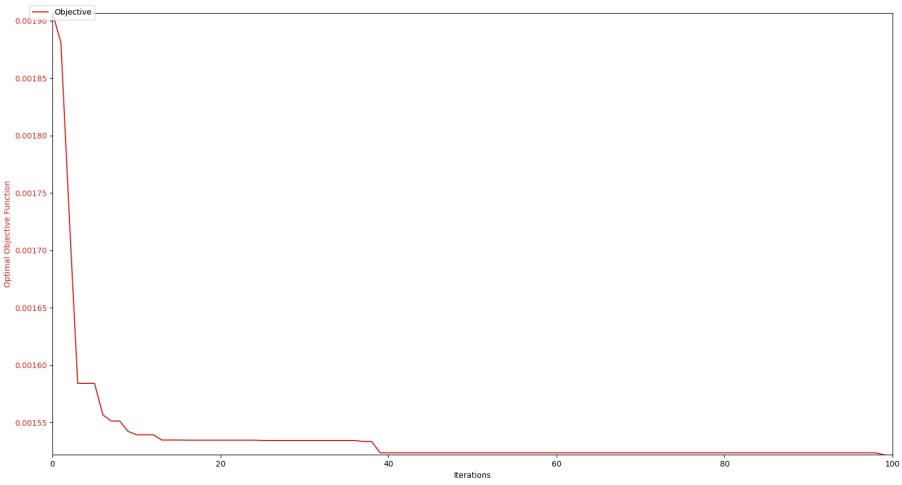

Figure 6.7: Best Fitness Over Generations in Case Study 3

For this case, It performed worse than all of the previous algorithms. We notice that swarm type algorithms generally don't perform well when increasing the problem size. This could be attributed to the weakening of the explorative power because they consider the solutions as positions in the grid. Therefore it becomes harder to cover more parts of the grid due to the nature of the method of exploration.

6.5 Case Study 4: 15x15 grid with Star Topology

- **Grid Size:** 15x15
- **Dead Cells:** [(2,2), (12,2), (2,12), (12,12)]
- **Population Size:** 50
- **Maximum Number of Generations:** 100
- **Inertia Weight:** 0.792
- **Cognitive Learning Factor:** 1.4944
- **Social Learning Factor:** 1.4944
- **Topology:** Star

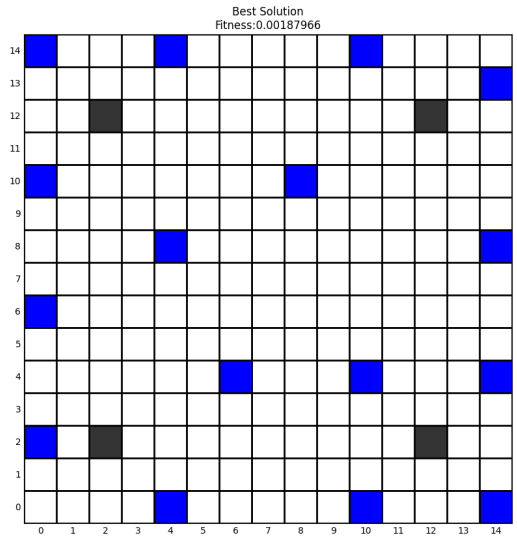

Figure 6.8: Best solution found by PSO in case study 4

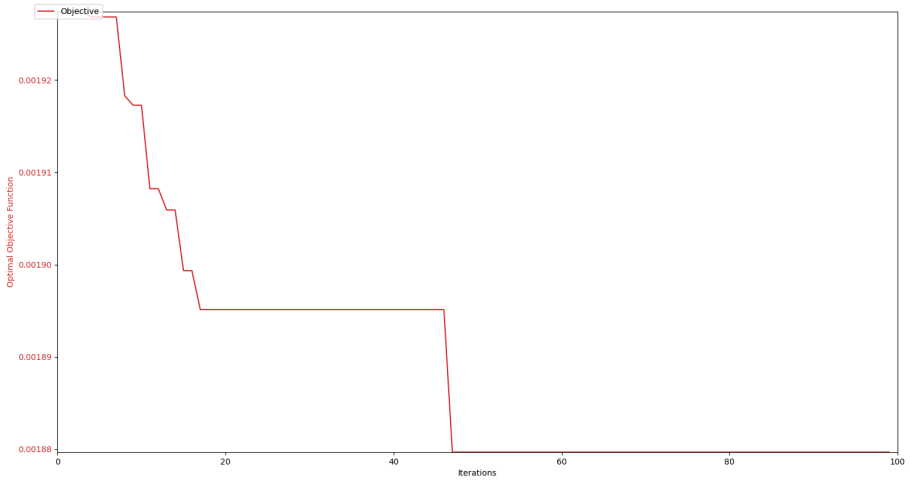

Figure 6.9: Best Fitness Over Generations in Case Study 4

For this case, we get a much better result than case 1. This is due to the new topology allowing each bird to be aware of the optimal bird. Due to the large local optima, the amount of birds that explore past it aren't many, meaning that the ring topology fails to notify all birds of this new optima. The star topology allows all the population to start moving towards this new optima leading to better results. This performs better than SA and GA.

6.6 Case Study 5: 20x20 grid with Star Topology

- **Grid Size:** 20x20
- **Dead Cells:** [(3,2),(4,2),(3,3),(4,3),(15,2),(16,2),(15,3),(16,3),(3,16),(4,16),(3,17),(4,17),(15,16),(16,16),(15,17),(16,17)]
- **Population Size:** 50
- **Maximum Number of Generations:** 100
- **Inertia Weight:** 0.792
- **Cognitive Learning Factor:** 1.4944
- **Social Learning Factor:** 1.4944
- **Topology:** Star

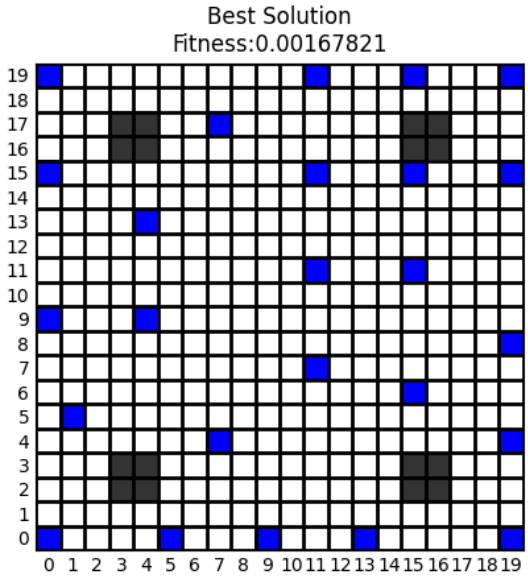

Figure 6.10: Best solution found by PSO in case study 5

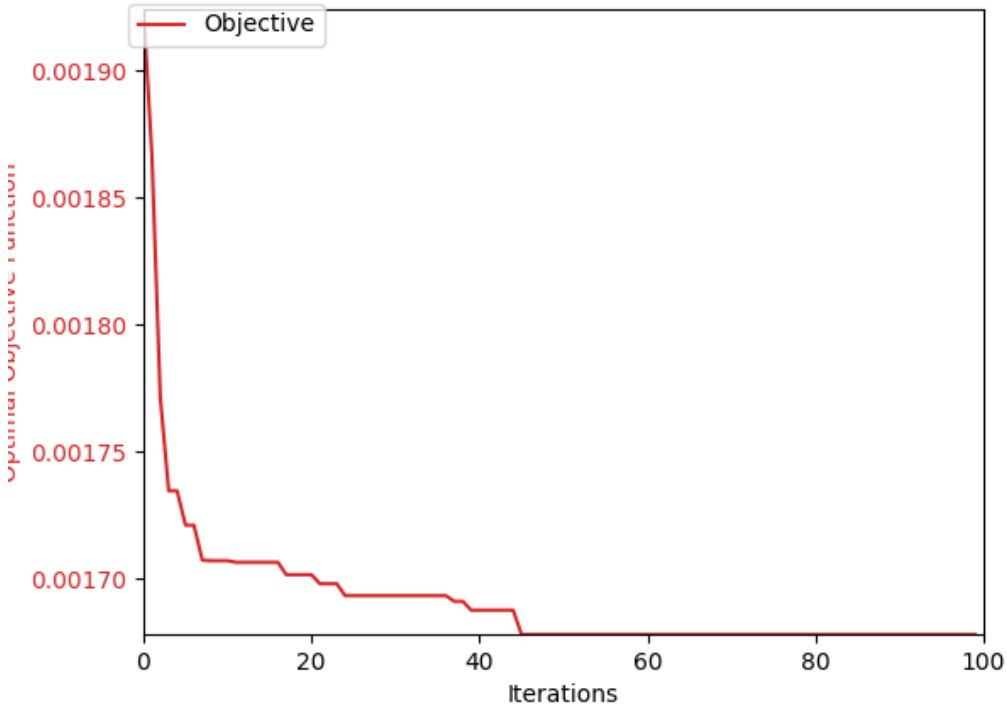

Figure 6.11: Best Fitness Over Generations in Case Study 5

For this case, we get a better result than case 2 and also outperform GA and SA.

6.7 Case Study 6: 25x25 grid with Star Topology

- **Grid Size:** 25x25
- **Dead Cells:** [(5,5),(5,6),(6,5),(6,6),(5,18),(5,19),(6,18),(6,19),(18,5),(19,5),(18,6), (19,6),(18,18),(18,19), (19,18),(19,19),(7,7),(7,6),(7,5),(7,18),(7,19),(18,7),(19,7) ,(5,7), (6,7),(5,17),(6,17),(7,17),(17,5),(17,6),(17,7),(17,17),(17,18),(17,19),(18,17),(19,17)]
- **Population Size:** 50
- **Maximum Number of Generations:** 100
- **Inertia Weight:** 0.792
- **Cognitive Learning Factor:** 1.4944
- **Social Learning Factor:** 1.4944
- **Topology:** Star

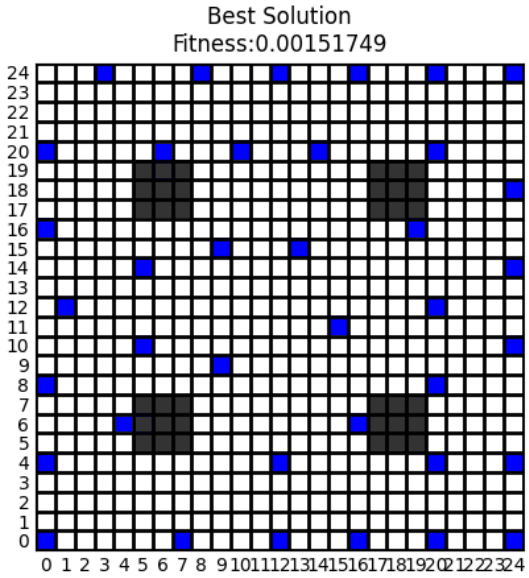

Figure 6.12: Best solution found by PSO in case study 6

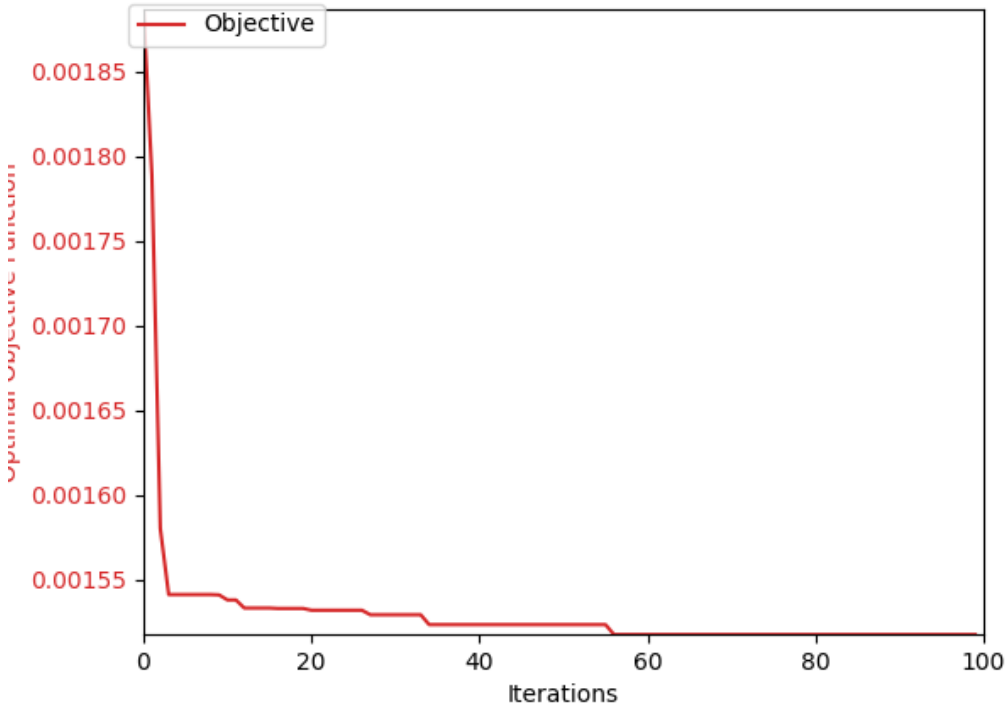

Figure 6.13: Best Fitness Over Generations in Case Study 6

For this case, we get a better result than case 3 and also perform better than GA and SA in

fitness.

6.8 Analysis of Wind farm Layout Optimization using Particle Swarm Optimization

In the analysis of the wind farm layout optimization using particle swarm optimization, it is clear that the topology used affects the convergence pattern and results. The following observations were made based on the six test cases, encompassing grid sizes of 15×15, 20×20, and 25×25 for both configurations:

Topology Impact

A trend can be noticed depending on the topology chosen. Star topology converges much quicker due to the global effect of each bird on the velocities. Ring topologies take longer or even may disregard new global minimas due to the neighbourhood size of 2. This has an impact on the results as ring topology has a chance of getting more easily stuck in local minima leading to no convergence as demonstrated in case 1. Ring topology is a much better for the problem as the global minima resides much farther than local minima requiring a higher degree of exploitation of potential solutions.

Illustrative Case: 25×25 Grid - Convergence

The difference in convergence rate across the two topologies:

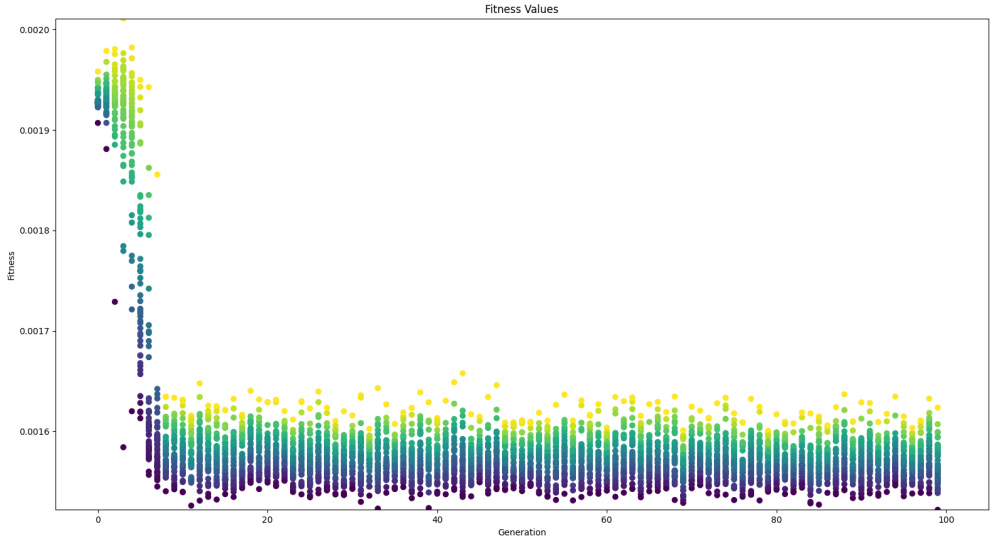

Figure 6.14: PSO population fitnesses against generations - Case 3

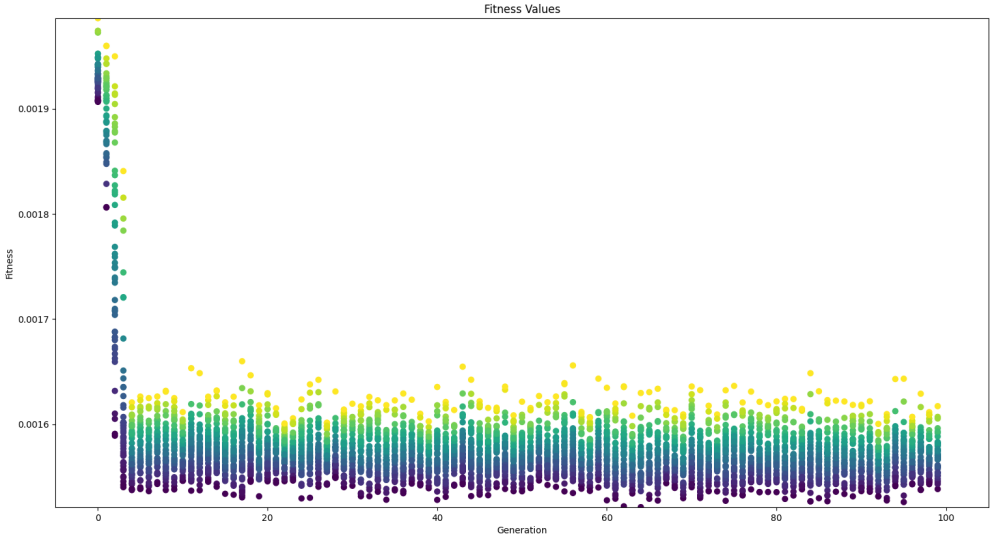

Figure 6.15: PSO population fitnesses against generations - Case 6

Star topology converges earlier due to the bigger effect of the global best bird in the swarm.

Illustrative Case: 25×25 Grid vs 20×20 Grid- Swarm Variance

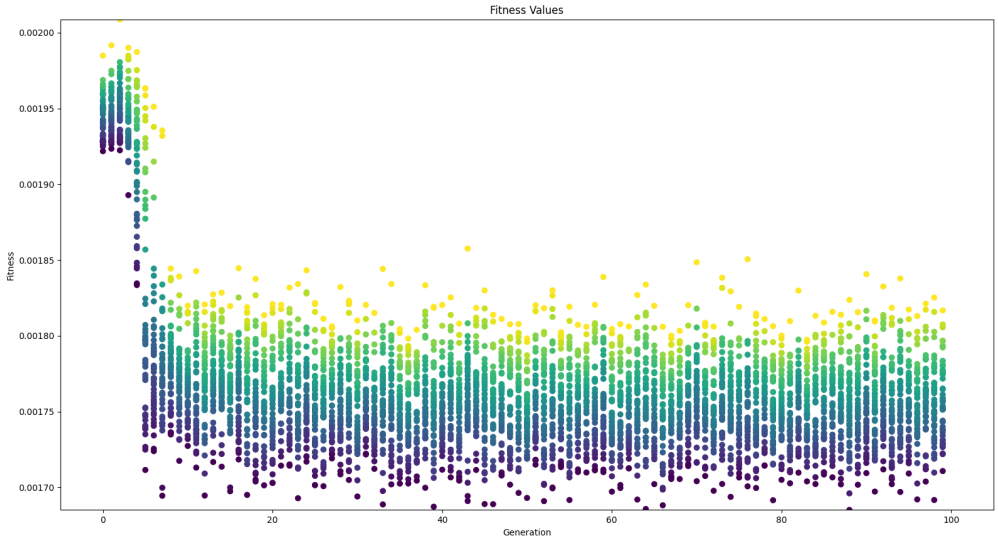

Figure 6.16: PSO population fitnesses against generations - Case 2

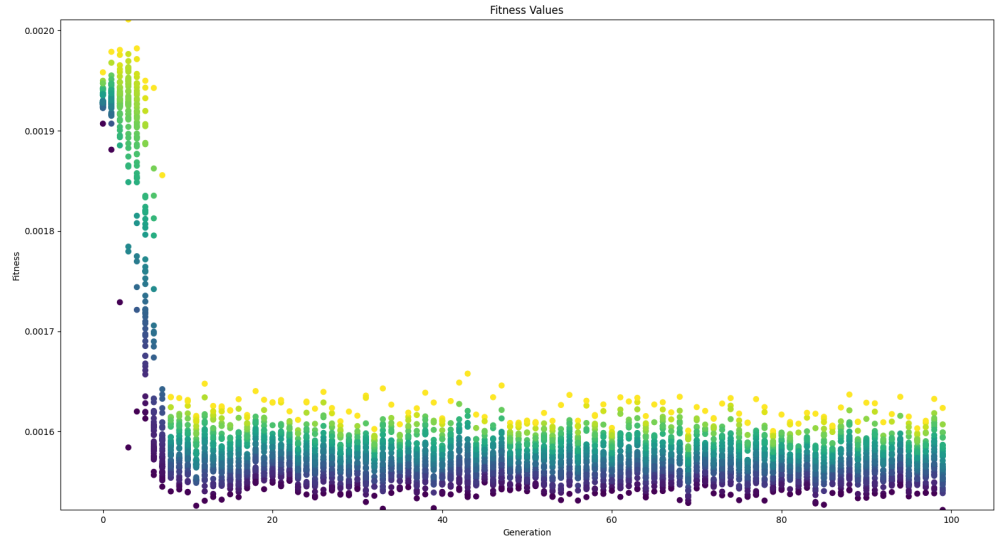

Figure 6.17: PSO population fitnesses against generations - Case 3

The swarm exhibits a much larger variance with many more outliers in smaller problem instances. The swarm occupies a larger range of values in the 20×20 case.

Chapter 7

Moth Flame Optimization

7.1 Algorithm

Moth Flame Optimization (MFO) is a nature-inspired optimization algorithm that draws inspiration from the behavior of moths attracted to flames. In this algorithm, potential solutions are represented as moths in a multidimensional search space. The optimization process involves the simulation of moth movements toward a flame, where the intensity of the flame corresponds to the quality of solutions. The movement of moths is inspired by the concept of transverse orientation, a behavior observed in real moths navigating in the presence of a light source. Transverse orientation is a phenomenon where moths use a fixed angle relative to a light source, such as the moon or a distant artificial light, to maintain a straight-line flight path. This behavior allows moths to keep the light source at a constant angle in their field of view, aiding them in maintaining a straight trajectory.

In the context of MFO, the transverse orientation concept is adapted to guide the movement of artificial moths within the solution space. Each artificial moth is associated with a specific position in the solution space, representing a potential solution to the optimization problem. The moths adjust their positions by considering the angle between their current position and the global best solution, which acts as the ”flame” attracting the moths.

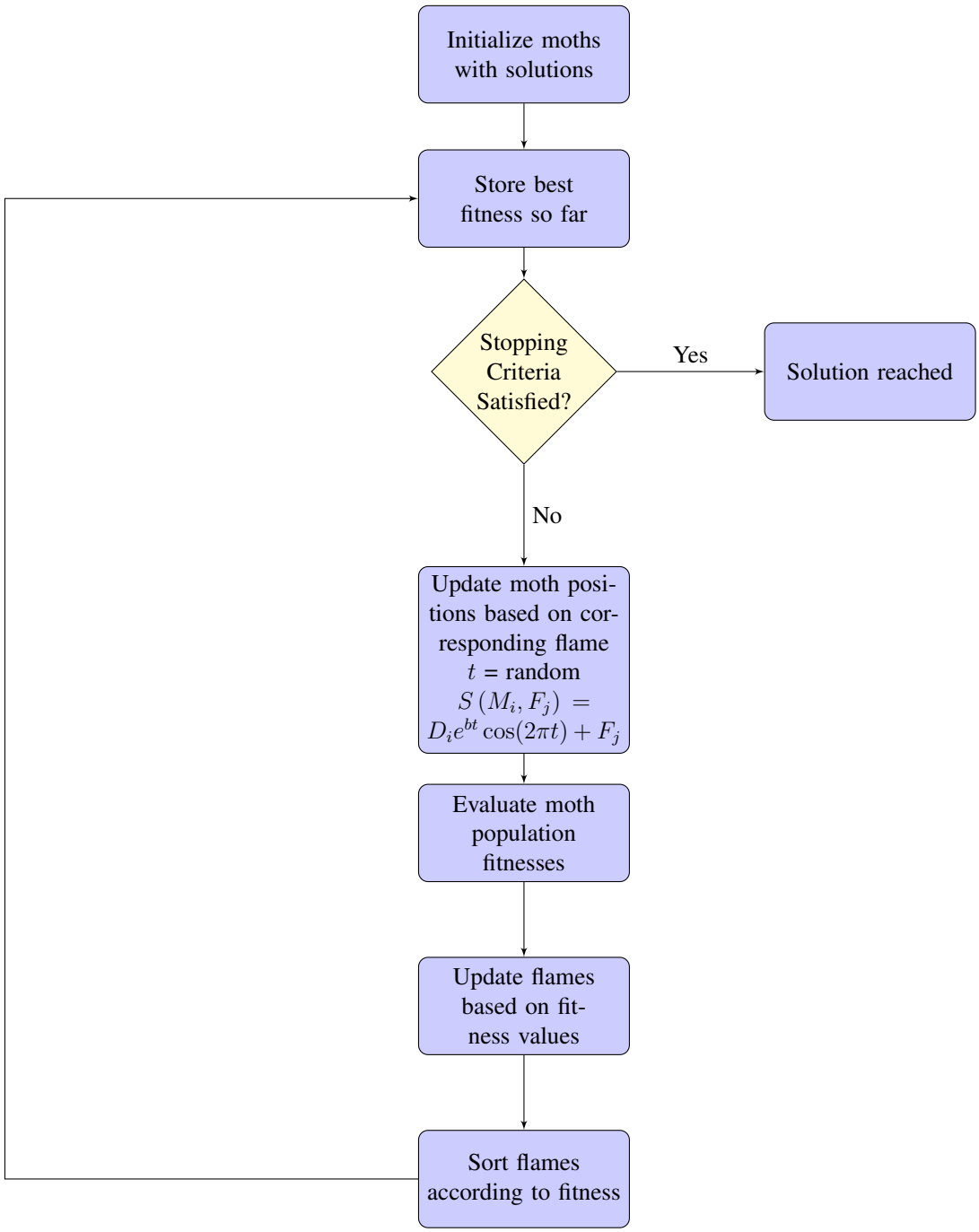

Figure 7.1: Flow chart of the Moth Flame Optimization algorithm

7.1.1 Initialization:

1. **Setting Parameters:** Begin by defining algorithm parameters, including the population size, spiral size coefficient, and range decay type.
2. **Initial Population:** Initialize a population of moths, where each moth represents a po-

tential solution. The position of each moth corresponds to a candidate solution in the search space.

7.1.2 Iterative Process

1. **Evaluation:** Assess the fitness of each moth based on the objective function. The fitness values guide the moths in their search for optimal solutions.
2. **Update Flames:** Adjust the flames based on the fitness values of the moths. Higher fitness values correspond to higher flames in the ordering. Flames are always kept sorted so that moths can be mapped to the flames.
3. **Moth Movement:** Simulate the movement of moths toward the flame based on their current positions:

$$S\left(M_i, F_j\right) = D_i \cdot e^{bt} \cdot \cos(2\pi t) + F_j \tag{7.1}$$

where Di is the distance of the ith moth for the jth flame, b is the spiral size coefficient, and t is a random number in the range [-1, 1]. The distance D_i is calculated as:

$$D_i = |F_j - M_i| \tag{7.2}$$

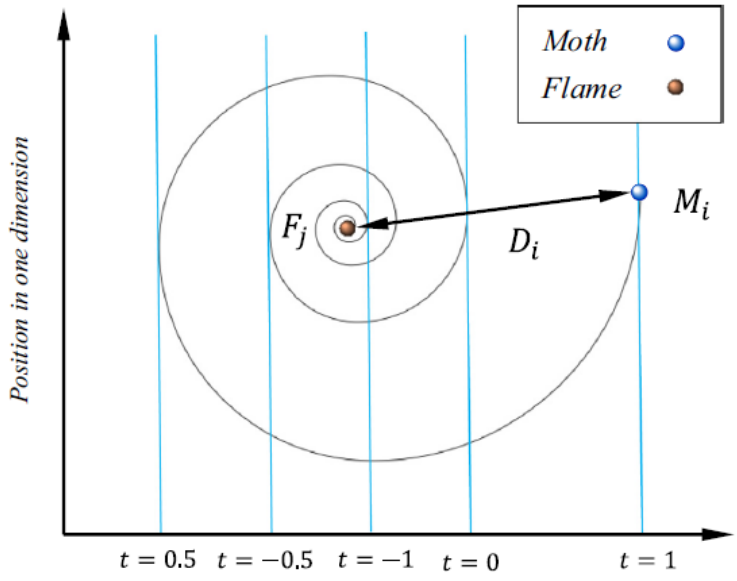

Figure 7.2: Moth distance to flame [3]

Due to the continuous nature of the MFO, we calculate the position of each turbine in a continuous space then discretize the values. We also calculate the delta in wind turbine

numbers between the moth and the flame using a discrete version of 7.1. If the moth has more turbines then the value gotten from the formula is the amount of turbines that are randomly removed. The same for the opposite case except turbines are added randomly instead of removed.

The lower bound of the value t is adaptively decreased to -2 based on the range decay type (linear, geometric) to incentivize exploitation at later iterations. This makes the spirals around the flames decay in size and thus limiting the exploration of each moth around the flame. The number of flames also decays with iterations to further increase convergence to the best flame:

$$flameno = \text{round}\left(N - l * \frac{N-1}{T}\right) \tag{7.3}$$

where l is the current number of iteration, N is the maximum number of flames, and T indicates the maximum number of iterations. With the number of flames decreasing every iteration, a mapping was devised to divide all the moths into evenly distributed moths across the flames until there is only 1 flame left. When this happens, all moths converge to that flame at the final iterations.

4. **Constraint Handling:** Ensure that the updated positions of moths adhere to any problem-specific constraints. If necessary, adjust the position of a moth to satisfy the constraints.
5. **Tracking the Best Solution:** Keep track of the best solution encountered throughout the iterations to preserve information about the optimal configuration.
6. **Termination:** The algorithm terminates after a specified number of iterations.

7.1.3 Conclusion

In the Moth Flame Optimization (MFO) algorithm, moths iteratively move toward a simulated flame, representing the optimal solution. The balance between exploration and exploitation is achieved by adjusting the flame intensity and guiding moths based on their fitness values. The algorithm's decay strategy influences the convergence behavior, making it adaptable to different optimization problems.

7.2 Case Study 1: 15x15 grid with Linear Decay

- **Grid Size:** 15x15
- **Dead Cells:** [(2,2), (12,2), (2,12), (12,12)]
- **Population Size:** 50

- **Maximum Number of Generations:** 100
- **Spiral Size Coefficient:** 0.3
- **Scheduling Function:** Linear Decay Function

$$t = t - \frac{1}{100}$$

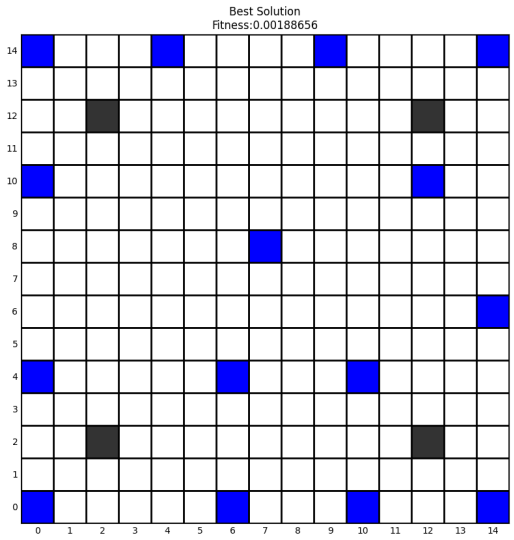

Figure 7.3: Best solution found by MFO in case study 1

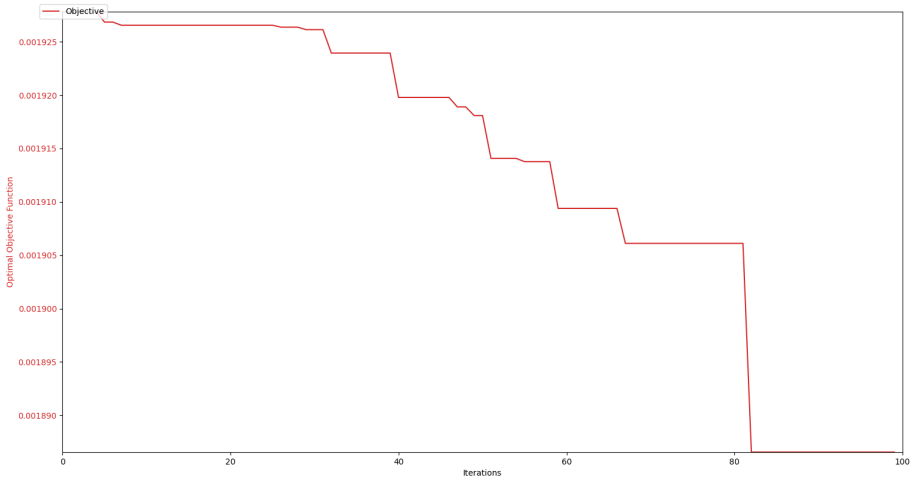

Figure 7.4: Best Fitness Over Generations in Case Study 1

This algorithm is much more robust and less susceptible to getting stuck in the local minimas of our problem. We score 0.001886 which is higher than both GA and PSO. It should be noted that the convergence occurred very late in the iterations due to the tendency of the algorithm to start exploitation at a late stage.

7.3 Case Study 2: 20x20 grid with Linear Decay

- **Grid Size:** 20x20
- **Dead Cells:** [(3,2),(4,2),(3,3),(4,3),(15,2),(16,2),(15,3),(16,3),(3,16),(4,16),(3,17),(4,17),(15,16),(16,16),(15,17),(16,17)]
- **Population Size:** 50
- **Maximum Number of Generations:** 100
- **Spiral Size Coefficient:** 0.3
- **Scheduling Function:** Linear Decay Function

$$t = t - \frac{1}{100}$$

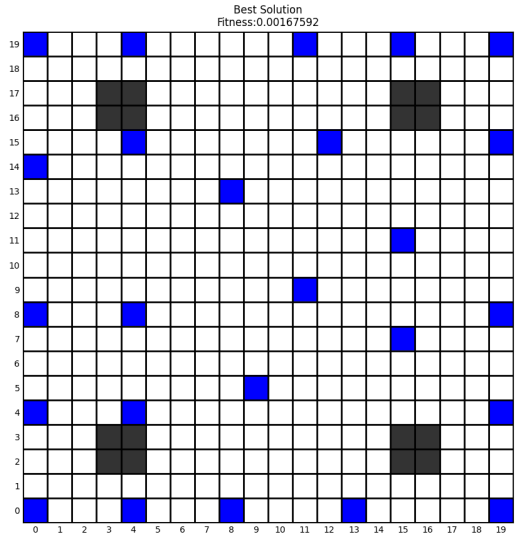

Figure 7.5: Best solution found by MFO in case study 2

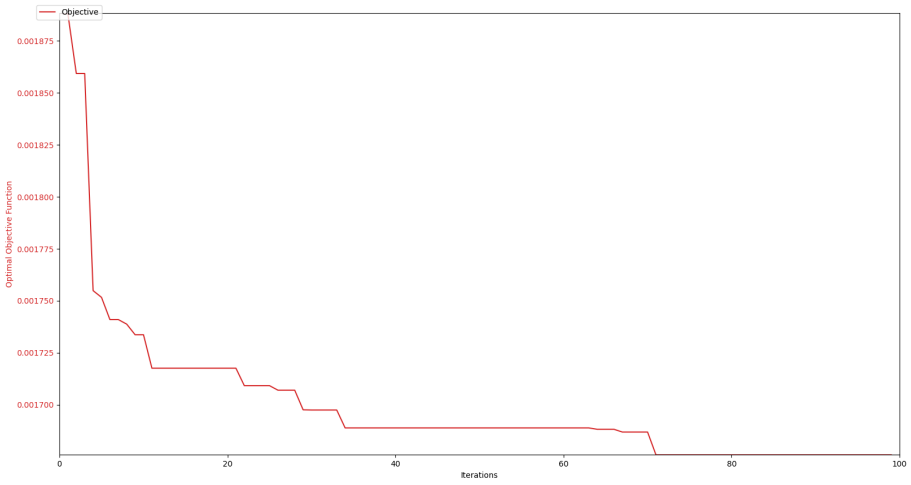

Figure 7.6: Best Fitness Over Generations in Case Study 2

For this case, the difference became smaller in the fitness. We scored the closest to GA with 0.001675 as opposed to 0.001670. We also overtook the performance of simulated annealing and PSO. Again a huge advantage of this algorithm is the fast execution time coupled with the high performance closing in on GA.

7.4 Case Study 3: 25x25 grid with Linear Decay

- **Grid Size:** 25x25
- **Dead Cells:** [(5,5),(5,6),(6,5),(6,6),(5,18),(5,19),(6,18),(6,19),(18,5),(19,5),(18,6) ,(19,6),(18,18),(18,19), (19,18),(19,19),(7,7),(7,6),(7,5),(7,18),(7,19),(18,7),(19,7) ,(5,7), (6,7),(5,17),(6,17),(7,17),(17,5),(17,6),(17,7),(17,17),(17,18),(17,19),(18,17),(19,17)]
- **Population Size:** 50
- **Maximum Number of Generations:** 100
- **Spiral Size Coefficient:** 0.3
- **Scheduling Function:** Linear Decay Function

$$t = t - \frac{1}{100}$$

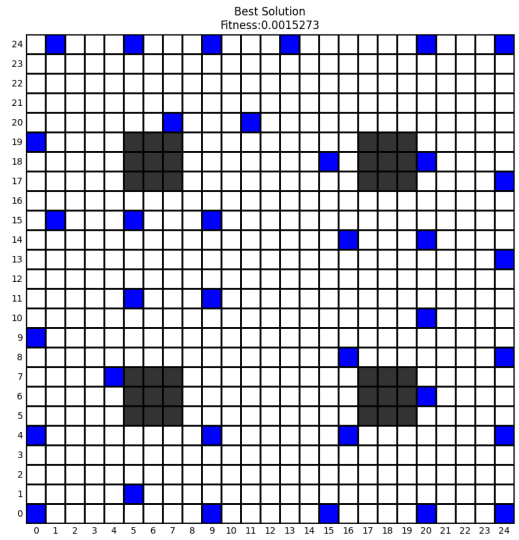

Figure 7.7: Best solution found by MFO in case study 3

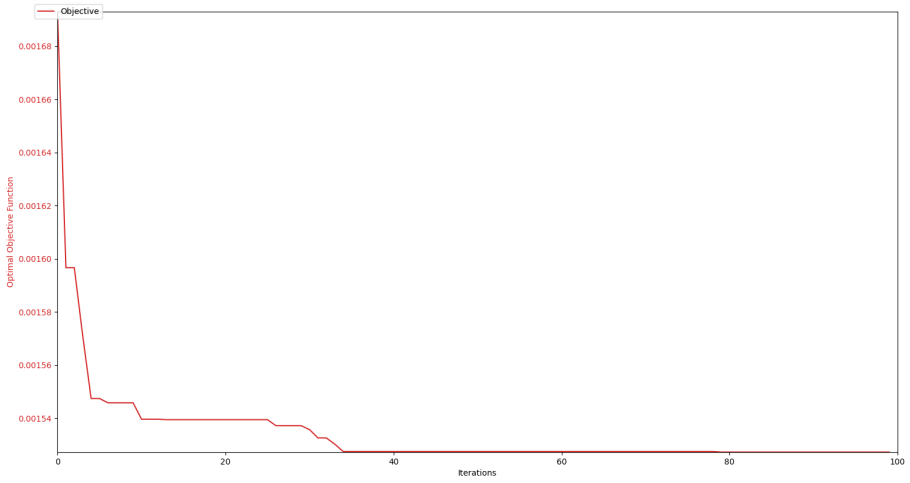

Figure 7.8: Best Fitness Over Generations in Case Study 3

For this case, our findings were surprising. It performed marginally worse than both SA and GA. The way PSO explores the search space according to the defined operators doesn't allow for as much mutation as GA and SA leading to it getting stuck in this minima. This is the only algorithm so far that doesn't exhibit better performance as the problem size scales.

7.5 Case Study 4: 15x15 grid with Geometric Decay

- **Grid Size:** 15x15
- **Dead Cells:** [(2,2), (12,2), (2,12), (12,12)]
- **Population Size:** 50
- **Maximum Number of Generations:** 100
- **Spiral Size Coefficient:** 0.3
- **Scheduling Function:** Geometric Decay Function

$$t = \sqrt[99]{2} \cdot t$$

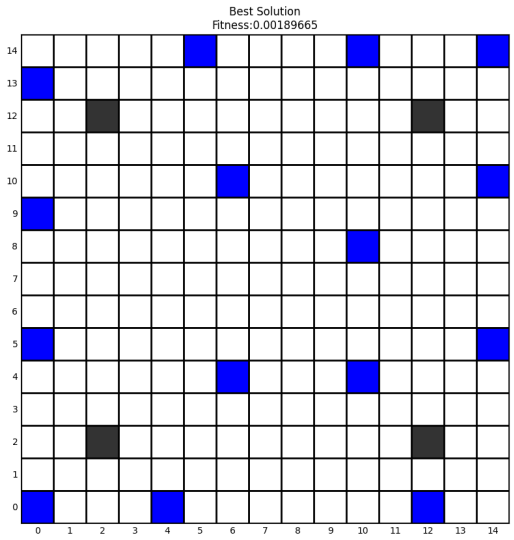

Figure 7.9: Best solution found by MFO in case study 4

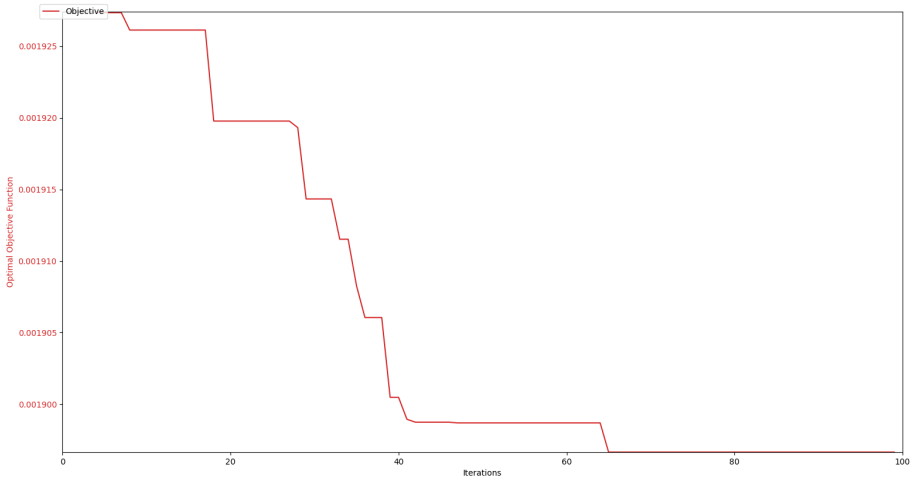

Figure 7.10: Best Fitness Over Generations in Case Study 4

For this case it performs marginally weaker. We attribute this to the fact that the 15x15 problem has many more local minimas which can easily make algorithms get stuck. For geometric decay the algorithm starts exploiting much earlier and thus limiting the exploration.

7.6 Case Study 5: 20x20 grid with Geometric Decay

- **Grid Size:** 20x20
- **Dead Cells:** [(3,2),(4,2),(3,3),(4,3),(15,2),(16,2),(15,3),(16,3),(3,16),(4,16),(3,17),(4,17),(15,16),(16,16),(15,17),(16,17)]
- **Population Size:** 50
- **Maximum Number of Generations:** 100
- **Spiral Size Coefficient:** 0.3
- **Scheduling Function:** Geometric Decay Function

$$t = \sqrt[99]{2} \cdot t$$

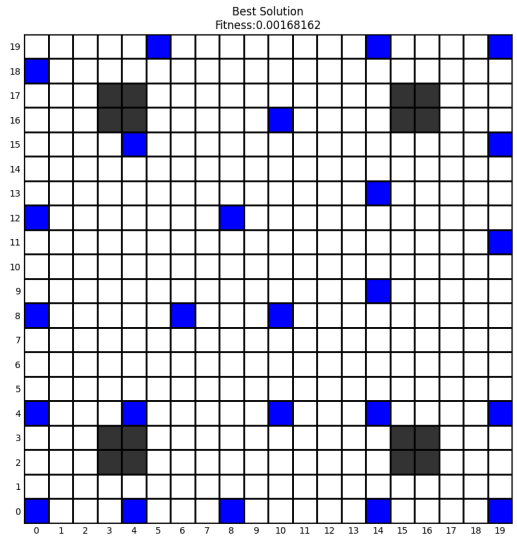

Figure 7.11: Best solution found by MFO in case study 5

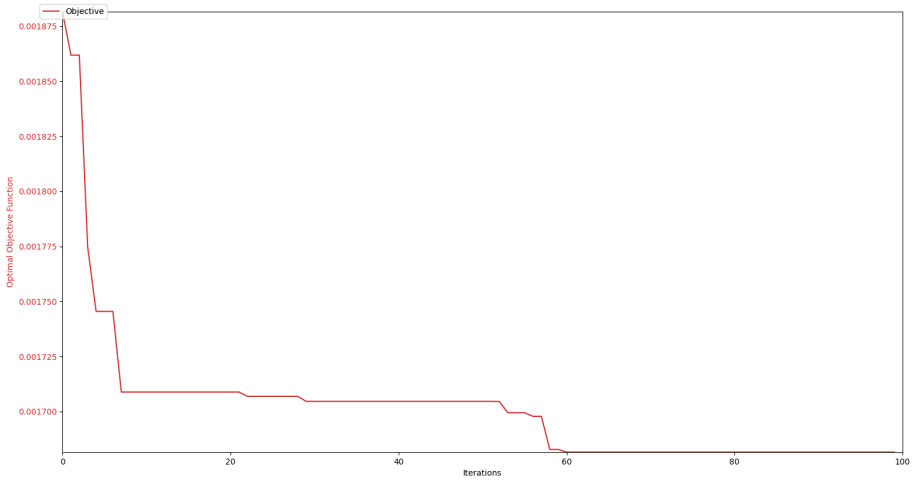

Figure 7.12: Best Fitness Over Generations in Case Study 5

For this case, we perform slightly worse due to the same reason.

7.7 Case Study 6: 25x25 grid with Geometric Decay

- **Grid Size:** 25x25

- **Dead Cells:** [(5,5),(5,6),(6,5),(6,6),(5,18),(5,19),(6,18),(6,19),(18,5),(19,5),(18,6), (19,6),(18,18),(18,19), (19,18),(19,19),(7,7),(7,6),(7,5),(7,18),(7,19),(18,7),(19,7) ,(5,7), (6,7),(5,17),(6,17),(7,17),(17,5),(17,6),(17,7),(17,17),(17,18),(17,19),(18,17),(19,17)]
- **Population Size:** 50
- **Maximum Number of Generations:** 100
- **Spiral Size Coefficient:** 0.3
- **Scheduling Function:** Geometric Decay Function

$$t = \sqrt[99]{2} \cdot t$$

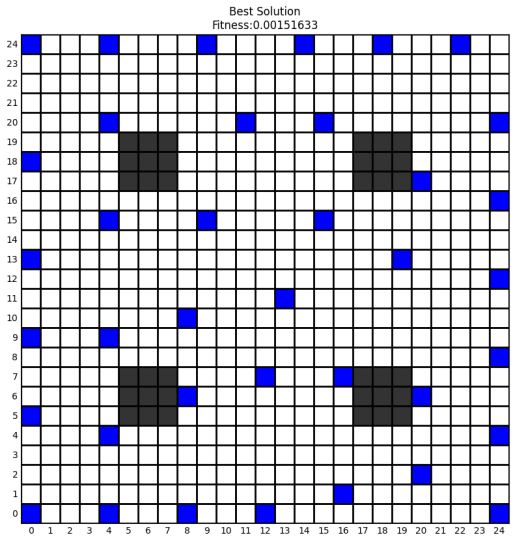

Figure 7.13: Best solution found by MFO in case study 6

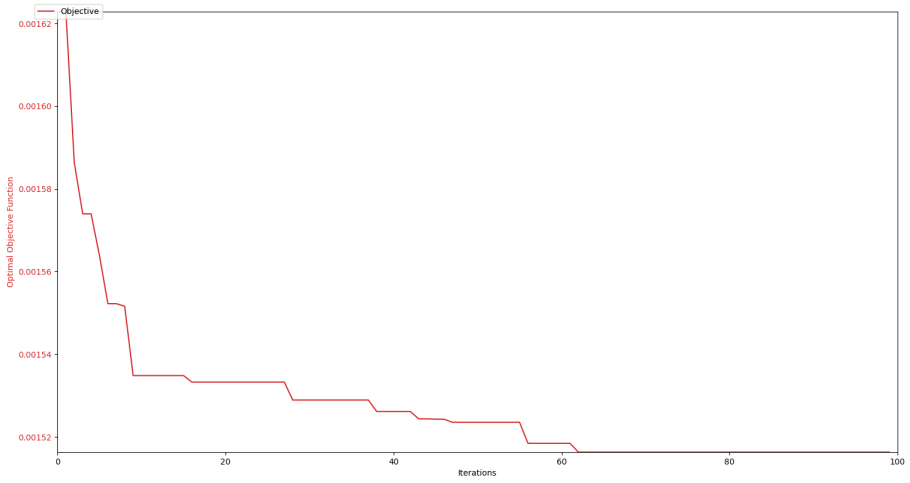

Figure 7.14: Best Fitness Over Generations in Case Study 6

This case is the outlier where it performs better than linear decay. We notice a trend that while the algorithm doesn't perform better than the other algorithms in the 25x25 case, geometric decay enhances its performance slightly. Still we surpass the performance of PSO making it a better choice for this size.

7.8 Analysis of Wind farm Layout Optimization using Moth Flame Optimization

In the analysis of the wind farm layout optimization using moth flame optimization, we analyze the algorithm with 2 decay approaches. The following observations were made based on the six test cases, encompassing grid sizes of 15×15, 20×20, and 25×25 for both configurations:

Decay Calculation Impact

As this algorithm has parallels to PSO and SA, we notice similar trends in it's behaviour. The decaying of the range of randomization for the spiral which causes the spiral to shrink causes exploitation at late iterations similar to SA. While in the earlier iterations, the swarm is left freely to explore the entire search space similar to PSO. The main advantage of MFO is it provides a balance between these aspects without sacrificing accuracy, while also having a respectable running time. Although it doesn't surpass the performance of GA, it is still a very effective approximation.

Illustrative Case: 20×20 Grid - Convergence vs other population-based algorithms

The difference in convergence rate across algorithms:

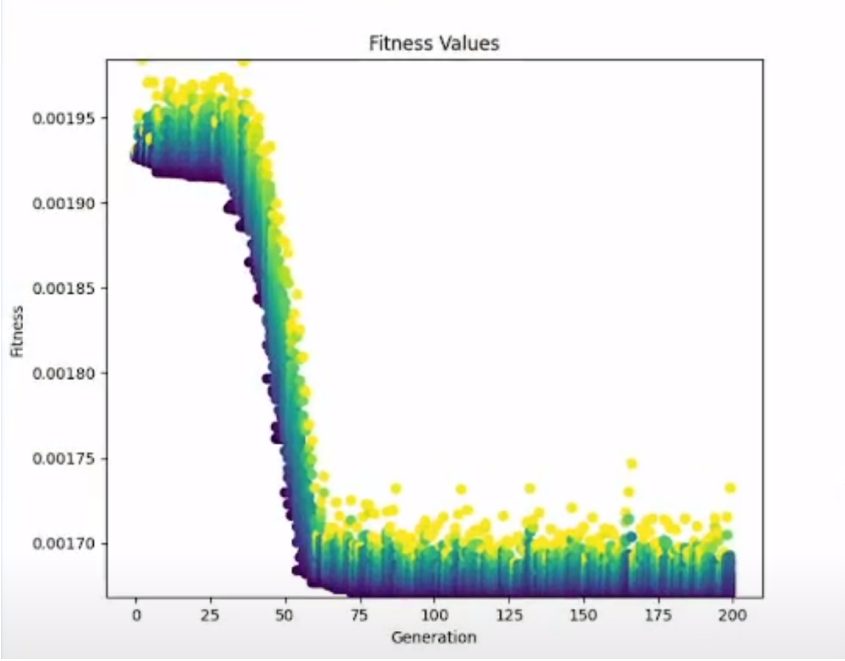

Figure 7.15: GA population fitnesses against generations

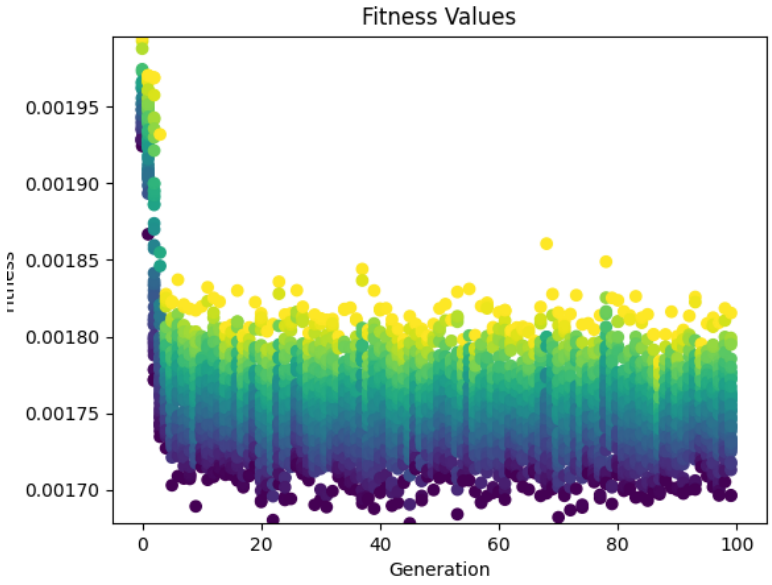

Figure 7.16: PSO population fitnesses against generations

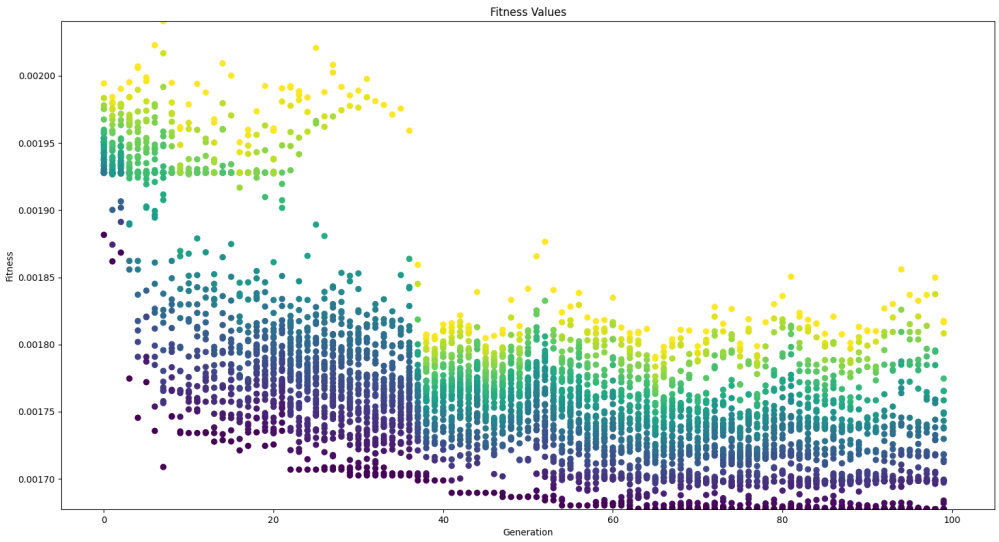

Figure 7.17: MFO population fitnesses against generations

The convergence behaviour of each algorithm varies wildly. GA Transitions smoothly during exploration and also exhibits very low variance after converging with very few outliers. PSO converges much earlier and more abruplty while maintaining high variance in the population. MFO is the most unpredictable with no clear convergence behaviour for the whole swarm except at late iterations. There is always high variance and even moths that get attracted to very

suboptimal flames as they haven't been updated yet creating wide gaps in the population.

Illustrative Case: Swarm Variance with problem size

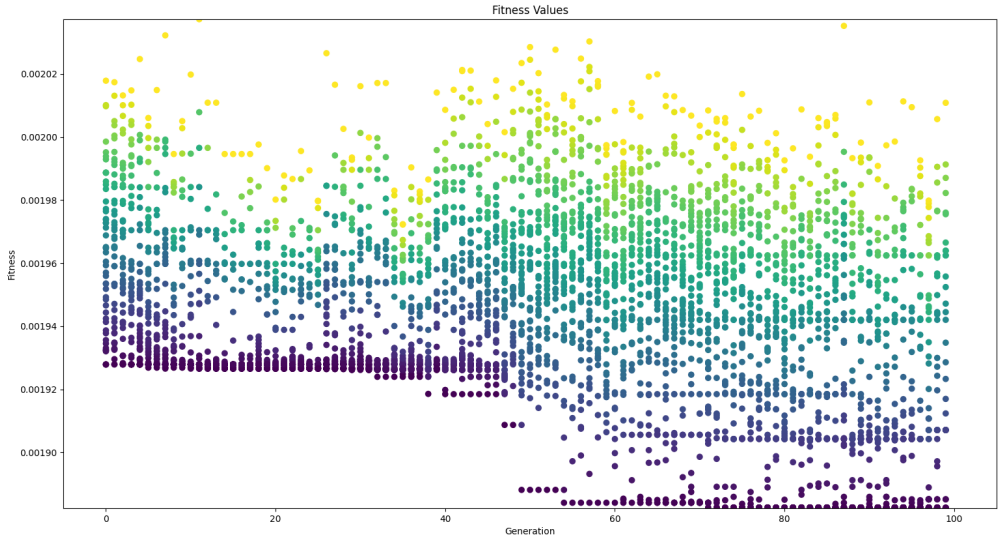

Figure 7.18: MFO population fitnesses against generations - Case 1

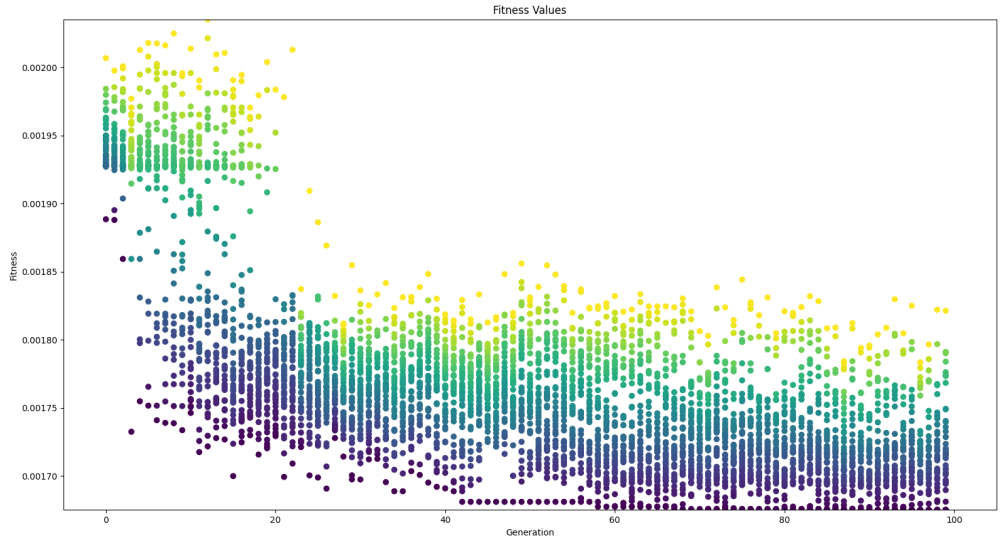

Figure 7.19: MFO population fitnesses against generations - Case 2

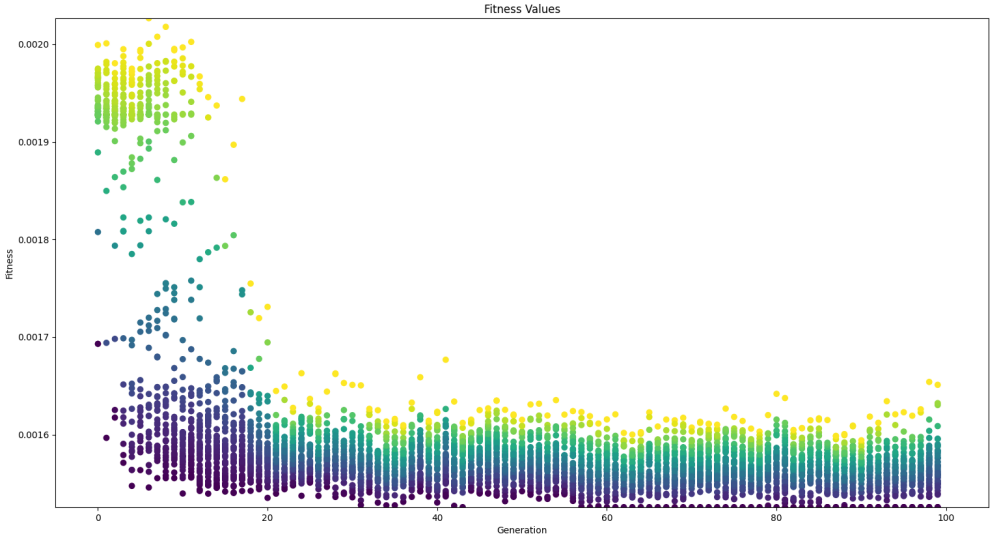

Figure 7.20: MFO population fitnesses against generations - Case 3

The swarm converges more uniformly as the problem size increases. This is evident especially in the 15x15 case. The size of the problem is too small such that it becomes too sensitive to any small changes in the grid. So each moth becomes more likely to be attracted to flames that made very small changes and thus caused big changes in fitnesses causing the swarm to diverge.

Chapter 8

Deep Q-Learning

Q-Learning is a classic RL algorithm that focuses on learning the value of taking specific actions in certain states. The Q-value (also known as the action-value) represents the expected cumulative reward an agent can obtain by taking a particular action from a given state and following a specific policy.

While traditional Q-Learning works well for small state-action spaces, it faces challenges when dealing with high-dimensional or continuous state spaces, as well as complex environments. Deep Q-Learning addresses these issues by employing a neural network to approximate the Q-values.

8.1 Model Architecture

In Deep Q-Network (DQN), a neural network is used to estimate Q-values. The neural network typically consists of one or more fully connected layers with activation functions to approximate the Q-function.

8.1.1 Replay Memory

To improve learning stability, DQN utilizes a replay memory buffer to store and sample past experiences. This memory buffer breaks the temporal correlation between consecutive samples, reducing the risk of the agent getting stuck in suboptimal policies (a local minima).

8.1.2 Exploration-Exploitation Strategy

DQN uses an exploration-exploitation strategy to balance exploration of new actions and exploitation of learned knowledge. This strategy helps the agent discover optimal policies while gradually reducing exploration as it becomes more confident in its Q-value estimates.

8.1.3 Training Process

The training process involves iteratively interacting with the environment, selecting actions based on the current state and exploration-exploitation strategy, and updating Q-values using the Bellman equation and the loss function for fitting the model.

The Bellman equation for Q-learning is typically represented as follows:

$$Q(s, a) = (1 - \alpha) \cdot Q(s, a) + \alpha \cdot [r + \gamma \cdot \max(Q(s', a'))] \tag{8.1}$$

Where:

- $Q(s, a)$ is the Q-value for taking action a in state s.
- α is the learning rate.
- r is the immediate reward received after taking action a in state s.
- γ is the discount factor.
- $\max(Q(s', a'))$ represents the maximum Q-value for the next state s' over all possible actions a'.

The mean squared error (MSE) loss function for Deep Q-Learning can be defined as follows:

$$L(\theta) = E\left[(Q(s, a; \theta) - (r + \gamma \cdot \max(Q(s', a'; \theta_{\text{target}})))^2\right] \tag{8.2}$$

where:

- $L(\theta)$ is the loss function dependent on the parameters θ.
- $E[\cdot]$ denotes the expected value over the distribution of experiences (state-action-reward-next state tuples).
- $Q(s, a; \theta)$ is the predicted Q-value for taking action a in state s given parameters θ.
- r is the immediate reward received after taking action a in state s.
- γ is the discount factor.
- $\max(Q(s', a'; \theta_{\text{target}}))$ represents the maximum Q-value for the next state s' over all possible actions a', calculated using a separate target network with parameters θ_{target}.

In summary The agent aims to learn a Q-function that accurately estimates the expected future rewards for each action.

Once trained, the DQN model can be evaluated in the target environment to assess its effectiveness in solving the problem it was designed for.

8.2 Environment

We model the environment using Gymnasium (previously OpenAI Gym). Gymnasium is a toolkit for developing and comparing reinforcement learning algorithms. It provides a collection of environments with a standardized interface. There are 3 main components to our environment that we implemented.

8.2.1 State

We model the current state of the environment as a grid of size nxm. We update the current state of the environment based on the actions taken by the agent. We reset the environment whenever after a specified number of training iterations are reached. We also model all of the constraints of the problem within the environment. To do so, every cell in the grid can take one of 4 values: 0 for cells available for adding turbines, 1 for cells containing turbines, 2 for cells that don't contain turbines but no turbines can be added to them due to spacing constraints, and 3 for dead cells.

8.2.2 Action

The actions available to the agent are either removing or adding new turbines to the grid while respecting all constraints. We also prevent repeated addition of the same turbines by marking any cell where a wind turbine was added and then removed as a dead cell. This prevents the scenario where the agent would keep adding and removing turbines at the same location repeatedly.

8.2.3 Reward

The reward of the agent is modeled as the delta in the objective function value from one state to another. This incentivizes the model to place in positions that have historically offered bigger deltas in fitness. As the model trains later in the iterations, it starts to assign bigger importance to these deltas, thus converging on the positions that best increase the fitness. When the action is adding a turbine, an additional 10 points are added. On the other hand, when the action is removing a turbine, a 10-point penalty is invoked. The additional incentive of +10 for adding a turbine (and the 10-point penalty when removing a turbine) is a strategic choice designed to encourage exploration of the search space with a higher number of turbines. This is because exploring a wider range of configurations often leads to better overall solutions. By incentivizing the addition of turbines, the system is more likely to explore various configurations, which might include a greater number of turbines, to identify the most efficient or effective layout. This exploration is crucial for avoiding local optima—situations where a configuration seems best within a limited set of options but is not the best overall. The reward function is described mathematically as:

$$\text{reward} = \begin{cases} (\text{old fitness value} - \text{new fitness value}) \times 10000 + 10, & \text{if action is adding a turbine} \\ (\text{old fitness value} - \text{new fitness value}) \times 10000 - 10, & \text{if action is removing a turbine} \\ 0, & \text{if action is invalid} \end{cases} \tag{8.3}$$

8.3 Configuration

The DQN agent is initialized with a neural network model using Keras. It comprises three fully connected layers, each employing ReLU activation functions. The input layer has 24 neurons, matching the state size, while two hidden layers also consist of 24 neurons each. The output layer contains neurons equal to the action size, representing possible turbine placement actions. Training relies on the mean squared error loss and the Adam optimizer.

A replay memory is implemented as a buffer with a maximum capacity of 2000 samples. This buffer stores tuples of (state, action, reward, next state, done), capturing the agent's experiences during training.

For exploration-exploitation strategy, the exploration rate (ϵ) is set initially at 1.0, allowing extensive exploration of the environment. Over time, ϵ gradually decays, encouraging exploitation of learned knowledge, with a minimum exploration rate of 0.01.

The training process consists of iterative interactions with the environment over 150 episodes. At each time step, the agent selects an action based on the current state and an action mask, executes the chosen action, and observes the next state, reward, and episode termination. The resulting experience tuples are stored in the replay memory.

The action mask is employed to restrict the agent's actions to valid choices, ensuring it can only add turbines to suitable positions or remove turbines from occupied cells. This approach accelerates learning and leads to practical solutions while respecting wind farm layout constraints.

In summary, our model configurations are:

- **Model Architecture**:
  - Input Layer Neurons: 24
  - Hidden Layers Neurons (each): 24
  - Output Layer Neurons (action size): [Specify your action size]

- **Replay Memory**:
  - Maximum Capacity: 2000 samples
- **Exploration-Exploitation Strategy**:
  - Initial Exploration Rate (ϵ): 1.0
  - Minimum Exploration Rate: 0.01
  - Exploration Rate Decay: 0.99
- **Training Process**:
  - Learning Rate (Adam Optimizer): 0.001
  - Batch Size: 32
  - Discount Factor (γ): 0.95
- **Performance Evaluation**:
  - Exploration Rate during Evaluation: 0.0 (Zero for exploitation)

8.4 Case Study 1: 15x15 grid

- **Grid Size:** 15x15
- **Dead Cells:** [(2,2), (12,2), (2,12), (12,12)]
- Number of Training Episodes: 150
- Steps per Episode: 100

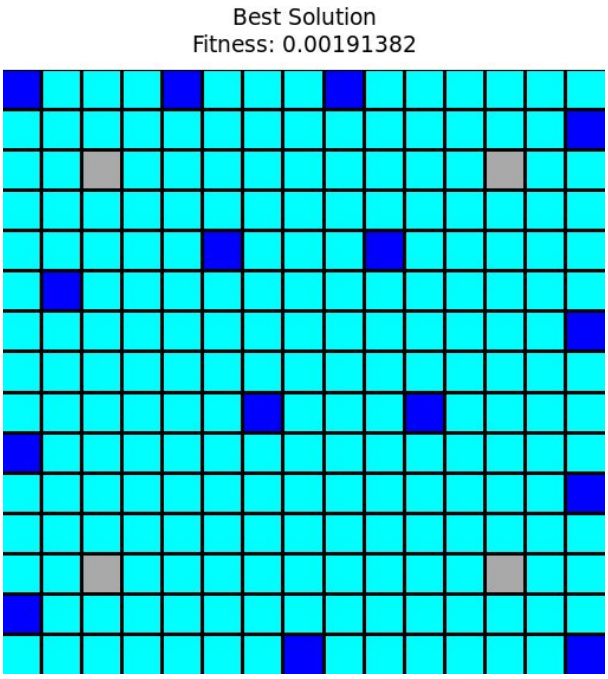

Figure 8.1: Best solution found by DQN in case study 1

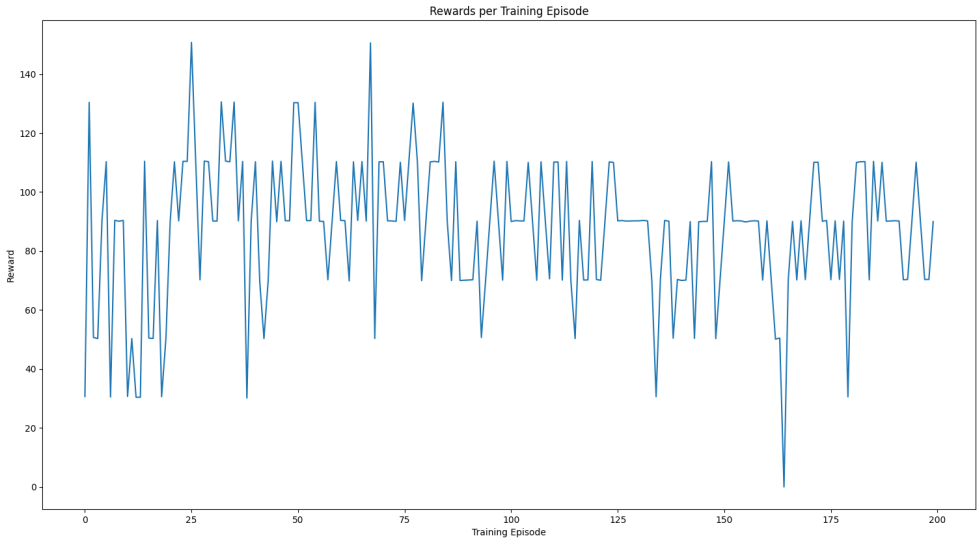

Figure 8.2: Reward over Episodes in case study 1

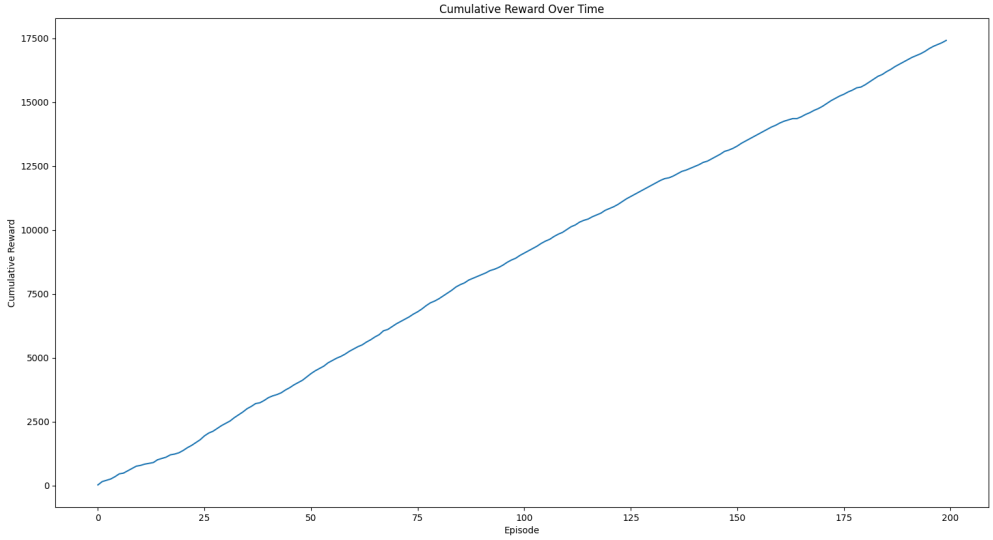

Figure 8.3: Cumulative Reward over Episodes in case study 1

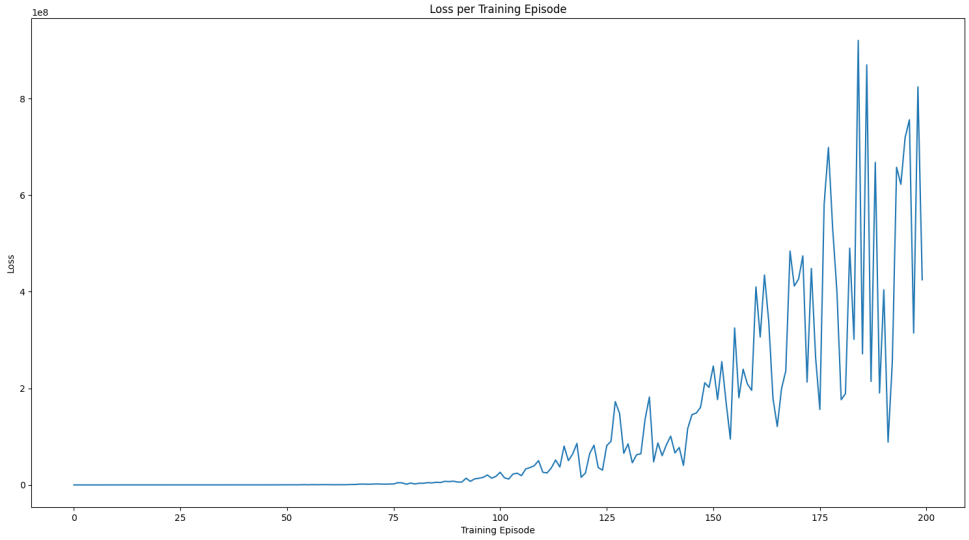

Figure 8.4: Loss over Episodes in case study 1

8.5 Case Study 2: 20x20 grid

- **Grid Size:** 20x20
- **Dead Cells:** [(3,2),(4,2),(3,3),(4,3),(15,2),(16,2),(15,3),(16,3),(3,16),(4,16),(3,17),(4,17),(15,16),(16,16),(15,17),(16,17)]
- Number of Training Episodes: 150
- Steps per Episode: 100

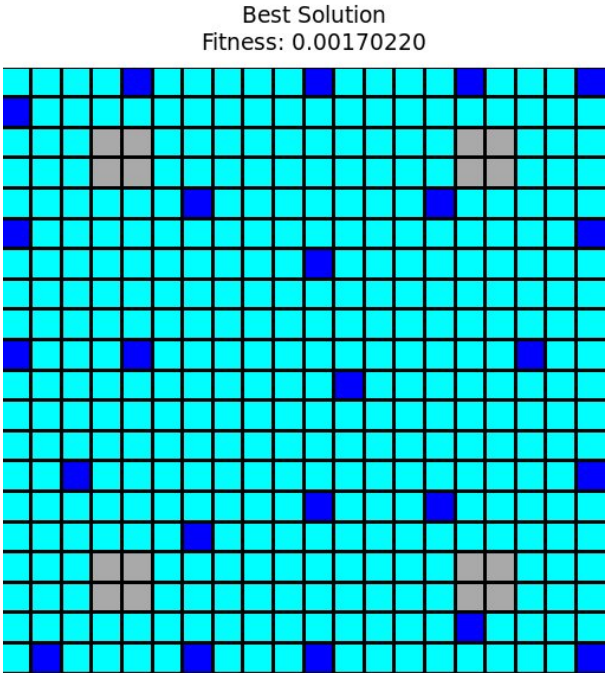

Figure 8.5: Best solution found by DQN in case study 2

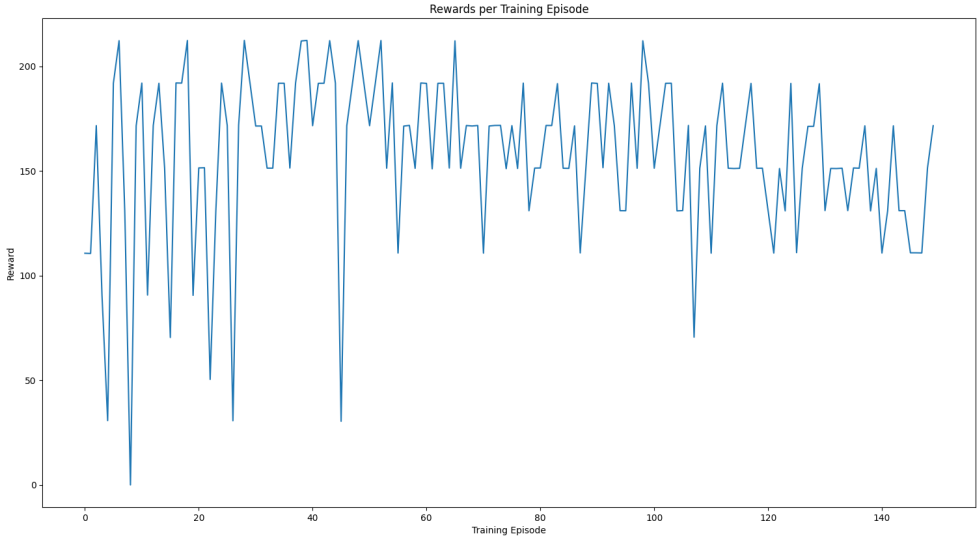

Figure 8.6: Reward over Episodes in case study 2

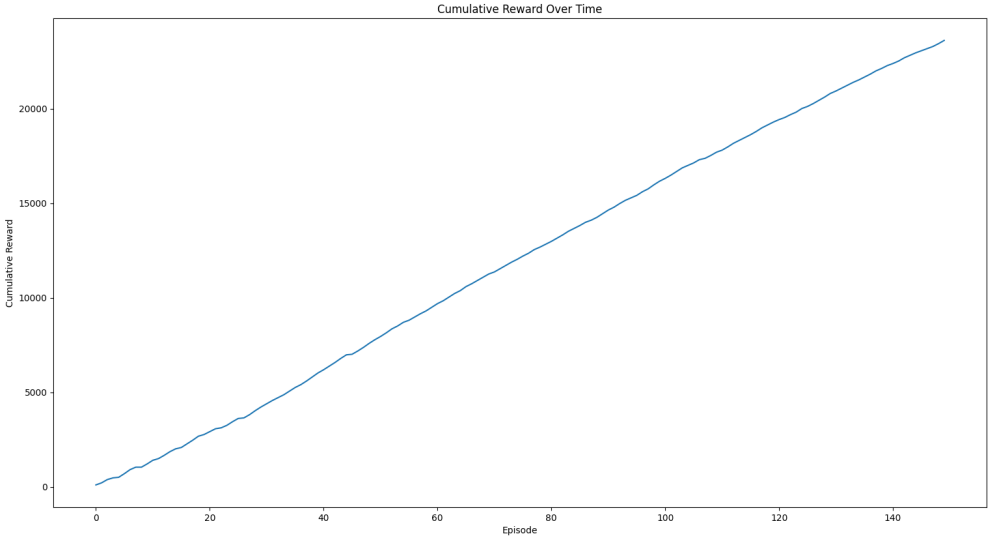

Figure 8.7: Cumulative Reward over Episodes in case study 2

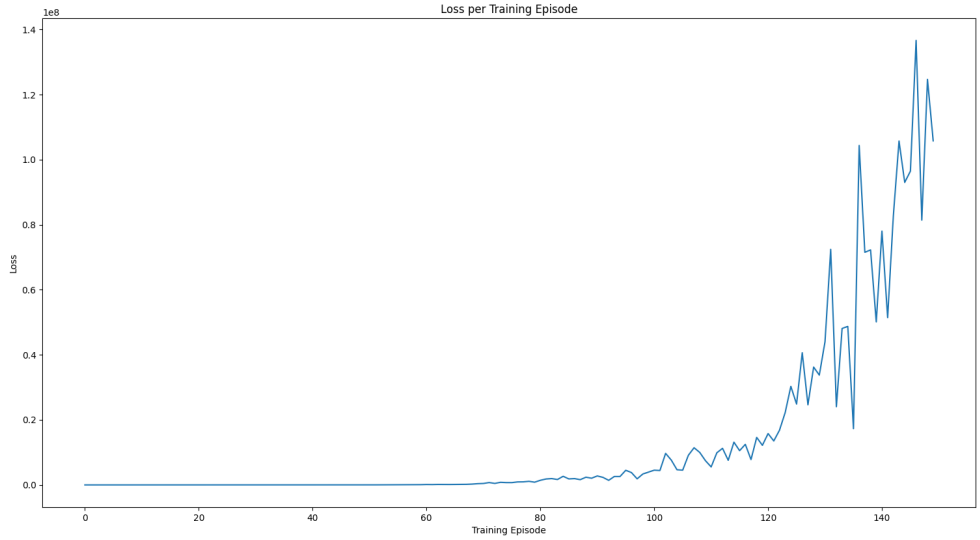

Figure 8.8: Loss over Episodes in case study 2

8.6 Case Study 3: 25x25 grid

- **Grid Size:** 25x25
- **Dead Cells:** [(5,5),(5,6),(6,5),(6,6),(5,18),(5,19),(6,18),(6,19),(18,5),(19,5),(18,6), (19,6),(18,18),(18,19), (19,18),(19,19),(7,7),(7,6),(7,5),(7,18),(7,19),(18,7),(19,7) ,(5,7), (6,7),(5,17),(6,17),(7,17),(17,5),(17,6),(17,7),(17,17),(17,18),(17,19),(18,17),(19,17)]
- Number of Training Episodes: 40
- Steps per Episode: 300

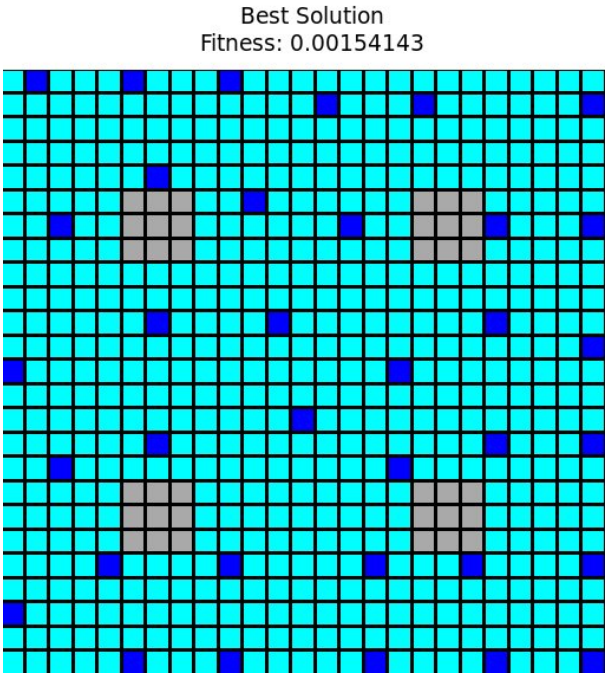

Figure 8.9: Best solution found by DQN in case study 3

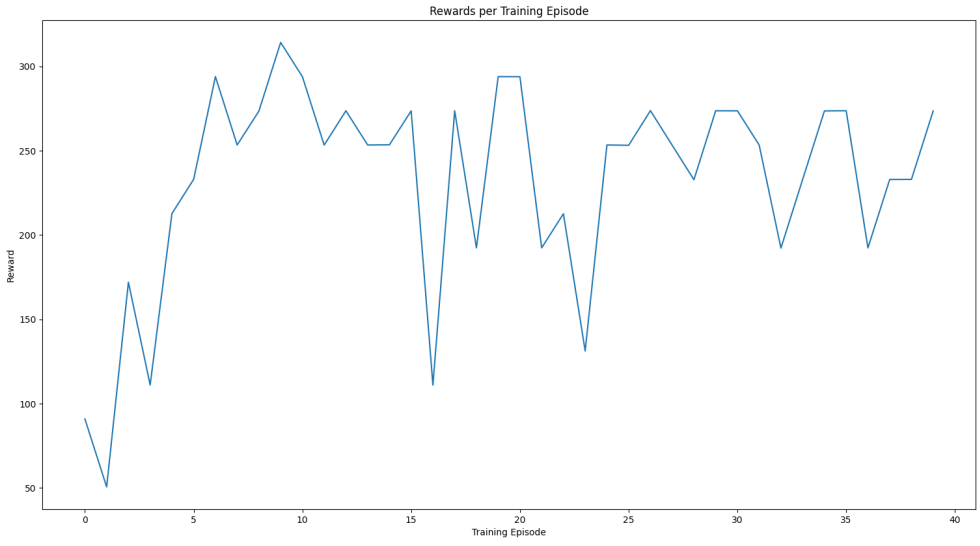

Figure 8.10: Reward over Episodes in case study 3

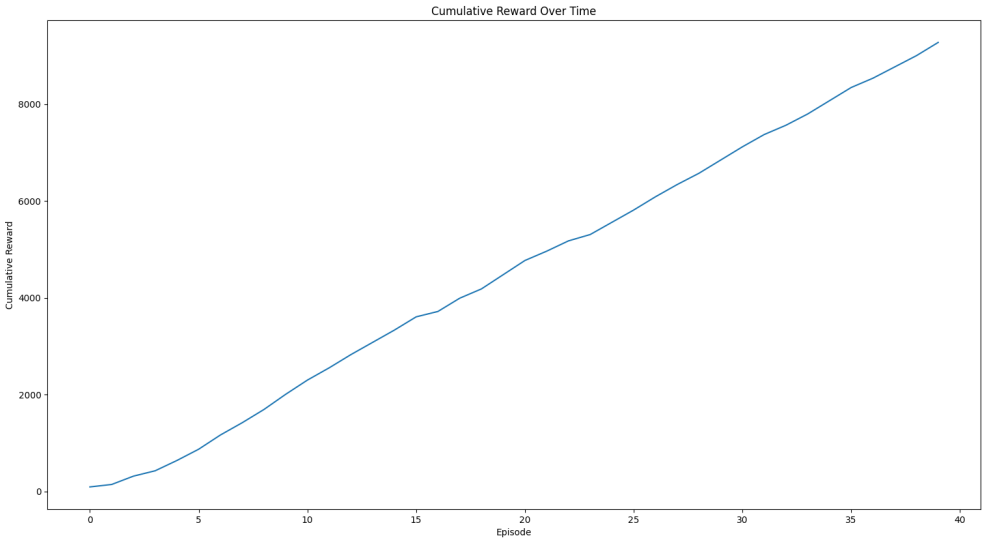

Figure 8.11: Cumulative Reward over Episodes in case study 3

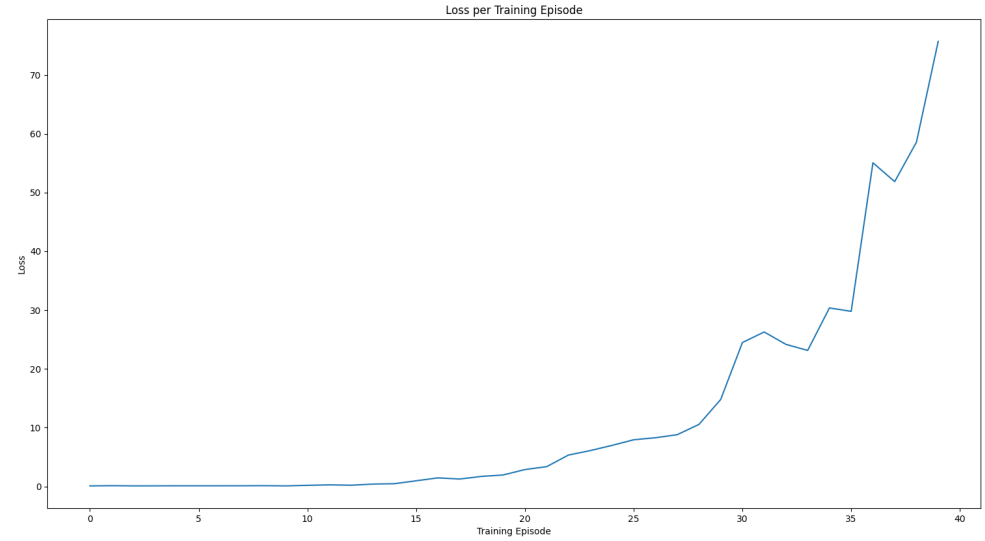

Figure 8.12: Loss over Episodes in case study 3

Chapter 9

Discussion and Conclusion

9.1 Data

Due to the stochastic nature of meta-heuristic algorithms, it will prove beneficial to not just study individual instances of algorithm performance, but also average results over multiple executions. We selected the highest performing configuration for each algorithm in each problem size, ran it 20 times and recorded the following metrics:

Algorithm	Problem Size	Best Fitness	Worst Fitness	Average Fitness	Std. Dev. of Fitness	Coefficient of Variation	Avg. Run Time (m)
Annealing case 1	(15x15)	0.001926	0.001927	0.001927	0.0000004	0.00022	0.41
GA Case 4	(15x15)	0.001870	0.001927	0.001788	0.0001131	0.06327	12.34
PSO Case 4	(15x15)	0.001871	0.001927	0.001892	0.0000169	0.00895	5.73
MFO Case 1	(15x15)	0.001871	0.001927	0.001895	0.0000196	0.01034	12.92
Annealing case 5	(20x20)	0.001760	0.001928	0.001870	0.0000596	0.03184	0.79
GA Case 2	(20x20)	0.001671	0.001920	0.001725	0.0000716	0.04154	11.01
PSO Case 5	(20x20)	0.001678	0.001695	0.001687	0.0000040	0.00238	23.18
MFO Case 2	(20x20)	0.001676	0.001694	0.001683	0.0000045	0.00265	36.53
Annealing case 6	(25x25)	0.001508	0.001528	0.001516	0.0000051	0.00337	47.38
GA Case 3	(25x25)	0.001495	0.001536	0.001508	0.0000089	0.00591	115.97
PSO Case 6	(25x25)	0.001514	0.001528	0.001522	0.0000040	0.00260	62.16
MFO Case 6	(25x25)	0.001512	0.001527	0.001520	0.0000044	0.00290	53.25

Table 9.1: Fitness and Run Time Statistics

Algorithm	Problem Size	Fitness Value	Maximum Reward	Minimum Reward	Average Reward	Total Reward
DQN Case 1	(15x15)	0.001914	151	0	87	17448
DQN Case 2	(20x20)	0.001702	212	0	157	23260
DQN Case 3	(25x25)	0.001541	314	51	232	9279

Table 9.2: Reward Statistics For the Deep-Q Learning Model

9.2 Discussion

9.2.1 Computational Complexity: Run Time Comparison

The run time is a crucial metric reflecting the computational efficiency of optimization algorithms. In this study, SA, GA, PSO, and MFO were all tested under different scenarios, each represented by a specific case. Notably, the average run times for SA cases (0.41, 0.793, and 47.38 minutes) were significantly lower than those for GA cases (12.34, 11.01, and 115.97 minutes). This suggests that, on average, SA exhibits superior computational efficiency compared to GA in the context of wind farm layout optimization. This is expected as GA is a population-based optimization technique which typically requires more time per iteration to generate a new generation and calculate its chromosomes' fitness values. The run times for PSO cases, also a population-based optimization algorithm, (5.73, 23.18, and 62.16 minutes) fall between those of Simulated Annealing (SA) and Genetic Algorithm (GA) across the different grid sizes. Notably, PSO demonstrates faster convergence than GA in all cases, showcasing its efficiency in calculating its solution. Compared to SA, PSO's run times are generally higher, but are close to each other in the 25x25 case. MFO's run times also lie close to PSO (12.92, 36.53, and 53.25 minutes) while offering comparable results.

The examination of run times also reveals an intriguing trend, particularly in the 25x25 case. In this scenario, all algorithms experience a substantial increase in run times compared to their counterparts in the 15x15 and 20x20 cases. For SA, the run time skyrockets to 47.38 minutes, for MFO to 53.25, for PSO to 62.16 minutes, and for GA to 115.97 minutes which is the largest by far. This considerable increase suggests that the optimization problem's complexity escalates significantly when dealing with a larger grid size, impacting the computational efficiency of the algorithms. MFO manages to be the fastest executing population based algorithm in our testing. It is crucial to acknowledge that these tests were conducted using multiple processing, which could have influenced the observed run time data. Parallel processing can impact the algorithms differently, potentially explaining variations in run times. While it contributes to faster computations overall, the intricacies of parallelization may also introduce some level of unpredictability in performance, especially in cases with larger grid sizes.

9.2.2 Optimality: Best Solution Comparison

The optimality of the solutions obtained by each algorithm is a crucial factor in assessing their effectiveness. Analyzing the best fitness values achieved in k runs for each case, it is observed that GA outperforms all the algorithms terms of optimality, despite the average longer run times. Overall, GA exhibits competitive or superior performance compared to others. The worst fitness values show a similar pattern, with PSO occasionally outperforming SA (as in 15x15 case: SA case 1 and PSO case 4), but generally falling short of GA's optimality. MFO stands in the middle of the spectrum with very balanced performance between GA and PSO while having better complexity than both.

The analysis of PSO results introduces a trade-off between optimality and computational efficiency. While GA remains superior in finding optimal solutions, PSO strikes a balance by offering competitive solutions with faster convergence and smaller run times. MFO provides the best tradeoff between run time and performance.

9.2.3 Repeatability: Mean and Standard Deviation Comparison

Repeatability is assessed through the mean and standard deviation of fitness values obtained in multiple runs. The mean fitness values for SA (ranging from 0.001516 to 0.001927) and GA (ranging from 0.001508 to 0.001788) show similar trends across different cases. However, it's essential to note that the standard deviation for SA (ranging from 0.0000004 to 0.0000596) is consistently lower than that for GA (ranging from 0.0000089 to 0.0001131). The coefficient of variation further highlights this difference, with SA having smaller values (ranging from 0.00022 to 0.03184) compared to GA (ranging from 0.00591 to 0.06327). This implies that SA tends to produce more consistent results, contributing to higher repeatability compared to GA.

PSO exhibits mean fitness values comparable to or slightly better than those of SA across different cases. Also, PSO's standard deviation values (ranging from 0.0000169 to 0.0000040) are lower than those of SA in 20x20 case and 25x25 case and higher in 15x15 case. The coefficient of variation also supports this observation. This means that PSO has higher stability in general than SA, with the variation in the 15x15 case explained by the better results of PSO. This implies that PSO explored more (and better) solutions with a higher number of wind turbines and more variations in the fitness values.

The comparison with GA reveals that PSO generally provides more consistent results, with lower standard deviation and coefficient of variation values. This aligns with the inherent exploration-exploitation balance of PSO, where particles collectively explore the solution space while being guided by the best-performing particles.

MFO follows closely with slightly higher deviations and variations across all cases. Despite this, it offers better average fitnesses in the 20x20 and 25x25 cases. The same can be said for the worst fitness in those cases which may make it more favourable in terms of reapeatability.

It has very good averages across the board only beaten by GA. Interestingly, it consistently has the best score in regards to worst fitness. This is useful when there is a lower bound on acceptable fitnesses to which MFO seems to guarantee.

While repeatability is a crucial metric in many optimization problems, its significance is somewhat diminished in the context of wind farm layout optimization. In this particular problem, stakeholders typically have the flexibility to run the optimization code multiple times before settling on a final layout. Given the stochastic nature of these optimization algorithms, the focus is often on obtaining diverse solutions and exploring the solution space comprehensively. As a result, the ability to replicate the exact solution across runs may be less critical, and the emphasis is instead placed on the variety of solutions generated and the optimality of the best outcomes. The iterative nature of wind farm layout optimization allows for a more relaxed consideration of repeatability, with decision-makers able to explore multiple runs to select the most suitable layout for their specific requirements.

9.2.4 Analysis of the Deep-Q Learning-Based Optimization

The DQN method's performance across the three problem sizes shows consistent learning, with cumulative rewards (Figures: 8.3, 8.7, 8.11) indicating continuous improvement in policy development. Despite the inherent variability in rewards per episode due to exploration, the DQN approach demonstrated the capacity to learn effectively within the environment and improved with each iteration. Although the reward per episode graph in the 15x15 case (Figure 8.2) shows no trend, the graphs for the other two cases (Figures: 8.6, 8.10) show better patterns with somewhat upward trend.

The loss per training episode graphs (Figures: 8.4, 8.8, 8.12), however, shows a more erratic pattern. Initially, the loss decreases, suggesting that the agent is learning. But as training progresses, the loss starts to fluctuate with some significant spikes. These fluctuations could be due to the learning rate being too high, causing the updates to overshoot the optimal policy parameters. Another explanation is the model complexity being enough to capture the policy for the environment but not too complex to generalize well.

Overall, The DQN agent's performance improved with the complexity of the problem, as evidenced by the cumulative rewards in all three scenarios. However, the increased variability in rewards and the upward trends in loss in the larger grid scenarios suggested that the agent's learning process was not as stable as it could be. This might be due to the complexity of the environment, potential overfitting, or the need for hyperparameter tuning.

The best fitness values achieved by the DQN method across the three problem sizes were competitive, particularly in the larger problem sizes. The DQN method's ability to maintain a balance between exploration and exploitation resulted in efficient and promising solutions within the constraints of the simulation environment. However, the solutions' fitnesses were

short of the solutions generated by the other optimization algorithms with also higher running time needed to train the model.

9.2.5 Conclusion and Recommendations

The comprehensive analysis, considering SA, GA, PSO, MFO, and DQN provides a holistic view of the strengths and weaknesses of each optimization algorithm in wind farm layout optimization. The choice between these algorithms depends on the specific priorities of the project.

If computational efficiency is paramount, especially in scenarios where run time is a critical factor, MFO emerges as a favorable option, offering faster convergence while maintaining competitive optimality. GA remains the top choice for achieving the most optimal solutions, albeit at the cost of longer run times. SA, while efficient in terms of run time, may fall short in optimality compared to population based techniques.

Ultimately, the selection of the optimization algorithm should align with the project requirements and the weight assigned to factors such as optimality, computational efficiency, and repeatability. Multiple runs with different algorithms can be leveraged to explore a diverse set of solutions, allowing stakeholders to make informed decisions based on the trade-offs presented by each algorithm.

In light of the findings and methodologies discussed in this paper, we propose the following future directions for research in the field of wind farm layout optimization:

1. **Advanced Optimization Algorithms:** Future research should explore the application of advanced or emerging optimization algorithms. This exploration could include cutting-edge metaheuristic algorithms or the development of algorithms specifically designed to address the unique challenges inherent in optimizing wind farm layouts.

2. **Hybrid Optimization Models:** There is significant potential in developing hybrid models that aggregate the strengths of various algorithms discussed in this study. For instance, a integration of Particle Swarm Optimization with Genetic Algorithms could potentially yield more robust and efficient solutions in wind farm layout optimization.

3. **Machine Learning Integration:** Integrating machine learning techniques with objective function calculations would greatly reduce the computational time. Machine learning algorithms could be utilized to predict the value given just the turbine setup greatly increasing the performance of the meta-heuristic algorithms and population based ones especially.

4. **Scalability Studies:** Conducting scalability studies is crucial to understand how the proposed optimization techniques perform as the scale and complexity of wind farms increase. Such studies would help in assessing the applicability of these optimization tech-

niques to larger, more complex wind farm projects. This is especially relevant due to the limitations of this study on problem size due to the available computational power.

Appendix

Project Github Repository

Bibliography

[1] Kyoungboo Yang and Kyungho Cho. Simulated annealing algorithm for wind farm layout optimization: A benchmark study. *Energies*, 12(23):4403, 2019.

[2] Tawatchai Kunakote, Numchoak Sabangban, Sumit Kumar, Ghanshyam G Tejani, Natee Panagant, Nantiwat Pholdee, Sujin Bureerat, and Ali R Yildiz. Comparative performance of twelve metaheuristics for wind farm layout optimisation. *Archives of Computational Methods in Engineering*, pages 1–14, 2022.

[3] Seyedali Mirjalili. Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *School of Information and Communication Technology, Griffith University, Nathan Campus, Brisbane, QLD 4111, Australia and Queensland Institute of Business and Technology, Mt Gravatt, Brisbane, QLD 4122, Australia*, pages 227–249, 2015. Received 23 May 2015; Received in revised form 25 June 2015; Accepted 11 July 2015; Available online 18 July 2015.

[4] Tarique Anwar Qureshi and Vilas Warudkar. Wind farm layout optimization through optimal wind turbine placement using a hybrid particle swarm optimization and genetic algorithm. *Environmental Science and Pollution Research*, pages 1–17, 2023.

[5] İbrahim Çelik, Ceyhun Yildiz, and Mustafa Şekkeli. Wind power plant layout optimization using particle swarm optimization. *Turkish Journal of Engineering*, 5(2):89–94, 2021.

[6] Xiawei Wu, Weihao Hu, Qi Huang, Cong Chen, Zhe Chen, and Frede Blaabjerg. Optimized placement of onshore wind farms considering topography. *Energies*, 12(15):2944, 2019.

[7] Md Amine Hassoine, Fouad Lahlou, Adnane Addaim, and Abdessalam Ait Madi. Wind farm layout optimization using real coded multi-population genetic algorithm. In *2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, pages 1–5, 2019.

[8] Maurice Clerc. Discrete particle swarm optimization, illustrated by the traveling salesman problem. 47, 01 2004.

[9] Xiaobing Yu and Yangchen Lu. Reinforcement learning-based multi-objective differential evolution for wind farm layout optimization. *Energy*, 284:129300, 2023.

[10] Hongyang Dong, Jingjie Xie, and Xiaowei Zhao. Wind farm control technologies: from classical control to reinforcement learning. *Progress in Energy*, 4(3), 2022.

[11] Giovanni Gualtieri. A novel method for wind farm layout optimization based on wind turbine selection. *Energy Conversion and Management*, 193:106–123, 2019.

[12] Niels Otto Jensen. *A note on wind generator interaction*, volume 2411. Citeseer, 1983.